МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Тольяттинский государственный университет»

Институт математики, физики и информационных технологий
(наименование института полностью)

Кафедра «Прикладная математика и информатика»
(наименование кафедры)

09.03.03 Прикладная информатика
(код и наименование направления подготовки, специальности)

ПРИКЛАДНАЯ ИНФОРМАТИКА В СОЦИАЛЬНОЙ СФЕРЕ
(направленность (профиль)/специализация)

БАКАЛАВРСКАЯ РАБОТА

на тему Разработка системы тестирования мобильных приложений

Студент Д.Е. Козлов _____
(И.О. Фамилия) (личная подпись)

Руководитель А.В. Шляпкин _____
(И.О. Фамилия) (личная подпись)

Допустить к защите

Заведующий кафедрой к.т.н., доцент, А.В. Очеповский _____
(ученая степень, звание, И.О. Фамилия) (личная подпись)

«_____» _____ 20_____ г.

Тольятти 2018

АННОТАЦИЯ

Актуальность темы - Разработка системы комплексного тестирования мобильного приложения по основным методикам, приоритетной задачей, тестирования является, выполнение анализа времени реакции сайта при многократном посещении, а так же определения скорости работы самого приложения. Актуальность обусловлена возможностью провести тестирование производительности, выявить наличие задержек в работе и определить скорость работы тестируемой системы. Объем выполненной бакалаврской работы – 78 страниц, содержит 29 рисунков и 14 таблиц, 20 литературных источников.

Объектом исследования – система тестирования мобильных приложений с выполнением комплексного тестирования.

Предметом исследования – процесс выполнения комплексного тестирования, с выполнением анализа задержек в работе сайта при многократном посещении и анализом скорости работы самой системы.

Цель – разработка системы тестирования мобильных приложений.

В введении описывается актуальность проводимого исследования, выделяются проблемы исследования, формируется цель и ставятся задачи.

В первой главе производится анализ видов и методов тестирования, существующих средств тестирования и условия выполнения тестирования.

Во второй главе проводятся обоснование выбора средства реализации и языка программирования, а так же описывается проектирование системы тестирования мобильных приложений.

В третьей главе описаны алгоритм выполнения действий, представлены изображения с комментариями, экранных форм реализованного программного продукт.

В заключении приводятся основные выводы, достигнутые в ходе выполнения работы.

Результатом работы является разработанная система для комплексного тестирования.

ANNOTATION

Actuality of the topic - Development of a system for integrated testing of a mobile application based on the main methods, a priority task, testing is, performing an analysis of the reaction time of the site during multiple visits, as well as determining the speed of the application itself. Relevance is due to the ability to conduct performance testing, identify the presence of delays in the work and determine the speed of the system under test. The volume of the bachelor's work is 78 pages, contains 29 figures and 14 tables, 20 literary sources.

The object of the study – system for testing mobile applications with complex testing.

The subject of the study – the process of performing complex testing, with the analysis of delays in the operation of the site during multiple visits and analysis of the speed of the system itself.

The goal – to develop a mobile application testing system.

The introduction describes the relevance of the ongoing research, highlight research problems, form the goal and set tasks.

The first chapter analyzes the types and methods of testing, the conditions for testing and existing testing tools.

In the second chapter, the rationale for choosing the architecture and designing the software product is carried out.

The third chapter describes the rationale for choosing the means of implementing an application that performs complex testing, presents images with comments, screen forms of the implemented software product.

In conclusion, the main conclusions reached in the course of the work are given.

The result of the work is the developed system for complex testing.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

На сегодняшний день практически у каждого человека есть мобильный телефон современного образца, который принято называть «смартфоном», это все тот же мобильный телефон, однако дополнен функциональностью карманного компьютера. В современных мобильных телефонах практически всегда имеются дополнительные функции, со временем разрабатываются и используются все более и более высокотехнологичные модели, для подчеркивания наиболее высокой функциональности и мощности вычислительных процессов, таких современных моделей для них был введен специальный определяющий термин «смартфон». В период роста популярности КПК – в этих устройствах, стали активно использовать функции мобильного телефона, вследствие чего такое устройство в дальнейшем получило название коммуникатор. В настоящее время разделение на смартфоны и коммуникаторы неактуально, оба термина обозначают одно и то же - миниатюрный универсальный компьютер с полноценными пользовательскими интерфейсами и развитыми радио-интерфейсами мобильного телефона. Современные смартфоны, в отличии от обычных мобильных телефонов, оснащены более развитой операционной системой, доступной для разработки и внедрения программных утилит со стороны других разработчиков. (операционная система, в простых мобильных телефонах, является недоступной для внедрения в нее сторонних программ от других разработчиков). Инсталляция дополнительных приложения позволяет в большой степени улучшить функциональность современных смартфонов, в отличии от обычных мобильных телефонов. Однако, у каждого приложения есть свои критерии качества, ведь приложение, будь оно установлено на телефон, является частью общей системы, а потому может серьёзно влиять на качество работы самого устройства. Приложение, которое является потенциально некорректным, может повлиять на такие функциональные характеристики как производительность, безопасность и простое «удобство пользования», нарушив привычный для пользователя интерфейс. Итак, перед тем как эти приложения

выйдут в свет, их тщательно тестируют. Необходимо понять, что же собой представляет это самое тестирование, как тестируются приложения и сколько всего видов тестирования есть на данный момент.

Объектом исследования – система тестирования мобильных приложений с выполнением комплексного тестирования.

Предметом исследования – процесс выполнения комплексного тестирования, с выполнением анализа задержек в работе сайта при многократном посещении и анализом скорости работы самой системы.

Цель работы – разработка системы тестирования мобильных приложений.

Для достижения поставленной цели необходимо решение следующих задач:

1. Проанализировать существующие виды и методы тестирования.
2. Проанализировать существующие системы тестирования.
3. Выполнить анализ возможной среды разработки мобильных приложений и языка программирования.
4. Выполнить анализ технологий разработки, систем тестирования мобильных приложений.
5. Спроектировать систему тестирования мобильных приложений.
6. Разработать систему тестирования на операционной системе Android для проведения комплексного тестирования.

Предмет исследования: Процесс выполнения комплексного тестирования.

Методы исследования: анализ моделирование, изучение источников по тематики.

ГЛАВА 1 АНАЛИЗ СИСТЕМ ТЕСТИРОВАНИЯ МОБИЛЬНЫХ ПРИЛОЖЕНИЙ

1.1 Анализ существующих систем тестирования мобильных приложений

На сегодняшний день существует множество систем тестирования, с помощью которых можно проверить поведение самых малых и незначительных частей, какой либо программы. Для тестирования самых разнородны проектов, зачастую применяется один и тот же инструментарий с возможностью выполнения типовых задач. Однако стоит учитывать, что тестирование происходит, основываясь на задачах тестирования и конкретной цели, в связи с этим можно заметить, что не все инструменты для тестирования одинаково применяются к различным методикам тестирования. Например, в тестировании нагрузки инструменты работают на уровне протокола, а для автоматизации регрессионного тестирования, инструменты работают на уровне графического использования интерфейсов. Для выявления и анализа проблем в различных частях системы есть достаточно большое количество инструментов.

При проведении тестировании необходимо конкретно определить цели и задачи тестирования: для каких целей тестируется приложение, какие задачи будут рассматриваться при тестировании и что в дальнейшем делать с полученной информацией. Тестирование необходимо для выявления несоответствий в работе приложения, оно является один из жизненных циклов при проектировании и разработки любого программного продукта. Поскольку поможет найти ошибки еще на стадии проектирования. Для этого были рассмотрены некоторые существующие системы тестирования, с целью определим их назначение и возможности применения для тестирования мобильных приложения, учитывая их применение в различных видах и методах тестирования.

Стоит заметить, что существующие средства тестирования мобильных приложений являются достаточно эффективными и часто применяемыми.

Некоторые инструменты для тестирования мобильных приложений представлены ниже в таблице 1.1.

Таблица 1.1 – Существующие системы тестирования

ПО	**Комментарии**
Monkey	Программа Monkey, одна из составляющих Android SDK, с помощью нее создается поток случайных действий пользователя. Является эмулятором псевдовременных потоков событий, таких как: клики, жесты, касания, события в работе системы и т.д.
MonkeyRunner	Инструмент MonkeyRunner, представляет API для написания программ, управляющие устройством-Android или эмулятором вне кода-Android. Есть возможность написания программы, которая устанавливает приложение или тестовый пакет, произведет запуск, отправит нажатия, сделает скриншоты интерфейса и сохранит их
Getevent and Sendevent	Эти программы дают возможность пользователю зафиксировать последовательность действий путем записи, а затем воспроизвести ее. Для работы с этими программами не нужно наличие права доступа Root
Robotium	Инструмент тестирует нативные и гибридных приложений, с возможностью записи тестов в режиме «черного-ящика». Описывает действия на уровне интерфейса с возможностью сделать тестовый скрипт который не зависит от размеров и расположения элементов интерфейса, разрешения и ориентации экрана. Есть возможность проверить реакцию тестируемого приложение, на какие либо действия, проверить состояние приложения после каждого шага и с помощью такой функции легок удается находить шаг в котором была допущена ошибка

ПО	Комментарии
Ranorex	GUI-Фреймворк предназначенный для автоматизации тестирования мобильных и web-приложений. У него нет собственного языка в нем используются C# и VB.Net. Позволяет осуществлять запись проводимых тестов и их результатов.
Espresso Test Recorder	Является инструментом для тестирования пользовательских интерфейсов приложений Android. В этом инструменте основной API достаточно мал и прост, но имеет открытый исходный код, что позволит его расширить и применить для своих нужд

На сегодняшний день существует достаточно большое число инструментов для разных методов тестирования. Однако, не смотря на большое количество доступных инструментов, при тестировании необходимо выбирать тот инструмент, который будет соответствовать задачам тестирования, опираясь на выбранную методику. Так же, стоит учитывать возможные недостатки существующих систем и пользоваться преимуществами. Для этого производилась анализ преимущественных сторон и недостатков описанных систем тестирования и представлен в таблице 1.2.

Таблица 1.2 – Преимущества и недостатки систем тестирования

ПО	Преимущества	Недостатки
Monkey	1. Отсутствие затрат на обслуживание. 2. Отсутствует зависимость от устройства. 3. Тестирование нагрузки позволит вывить сложные и непосредственные ошибки.	1. Для разных приложений может заметно отличаться качество тестирования 2. Достаточно тяжело воспроизвести последовательность действий, вызвавших ошибку 3. Во время тестирование не проверяется состояние приложения

ПО	Преимущества	Недостатки
MonkeyRunner	1. Гибкий инструментарий.	1. Достаточно сложное написание сценариев, даже для простых приложений.
Getevent and Sendevent	1. Шаги последовательности можно написать без лишних затрат, в ходе проведения ручного тестирования. 2. Для записи шагов последовательности событий не требуются навыки в области программирования.	1. Алгоритм действий подойдет только для одного устройства при использовании фиксированного экрана. 2. Последовательность шагов необходимо описывать отдельно для каждого устройства. 3. Во время теста не проверяется состояние приложения. 4. Для воспроизведения сложных и быстрых шагов последовательности действий уходит больше.
Robotium	1. Действия можно описывать на уровне пользовательского интерфейса. 2. Написанный скрипт не зависит от разрешения и ориентации экрана. 3. После действий скрипта проверяется состояние приложения. 4. Создаются сценарии высокого качества с разумными затратам	1. Достаточно сложное написание сценариев на языке Java. Это потребует значительное количество времени и навыки в области программирования. 2. Если меняется интерфейс в приложении, то необходимо изменять скрипт. 3. Создание тестовых сценариев сложнее, чем запись событий

ПО	Преимущества	Недостатки
Ranorex	1. Отсутствие собственного языка	1. Возможность записи проводимых тестов и их результатов
Espresso Test Recorder	1. Основной API достаточно мал и примитивен	2. Открытый исходный код, с возможностью расширения и применения под определенные действия.

На основе такого сравнительного анализа систем тестирования, можно выбирать оптимальный вариант, исходя из задач тестирования, возможностей и предпочтений. Используя определенны преимущества и избегая недостатков, можно проводить тестирование, с возможностью ограничиться от серьезных затруднений и выполнить необходимые задачи.

1.2 Назначение и процесс тестирования

Тестирование – это, одна из технологий проверки качества, оно осуществляется на стадии проектирования и в дальнейшем, перед выпуском разрабатываемого программного продукта. Основная задача тестирования обнаружить несоответствий в работе приложения, оно так же, является важным этапом жизненного цикла любого разрабатываемого приложения. Тестирование позволит обнаружить несоответствия между ожидаемым и реальным поведением программы, такая проверка осуществляется на наборе тестов, выбранных определенным образом. В тестирование включены определенные процессы такие как: проектирование тестов, выполнение тестирования и анализ полученных результатов.

Все эти процессы являются неотъемлемой частью всего процесса тестирования, где формируются тестовые случаи или сценарии, выполняется тестирования и анализируется поведение тестируемого приложения.

На рисунке 1.1 так же продемонстрирована логическая модель выполнения тестирования.

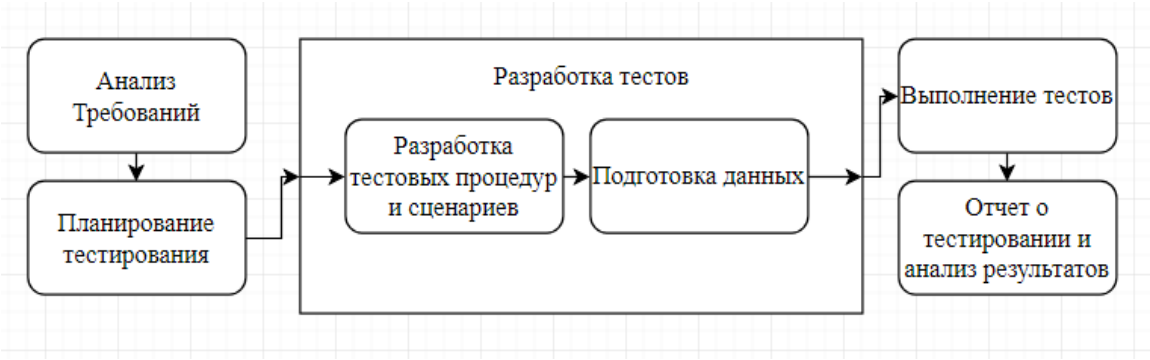


Рисунок 1.1 – Логическая модель тестирования

Если рассматривать в целом связь процессов тестирования и разработки программного продукта, можно выделить взаимосвязь, параллельно идущих процессов. На рисунки 1.2 продемонстрирована связь процессов тестирования и разработки ПО.

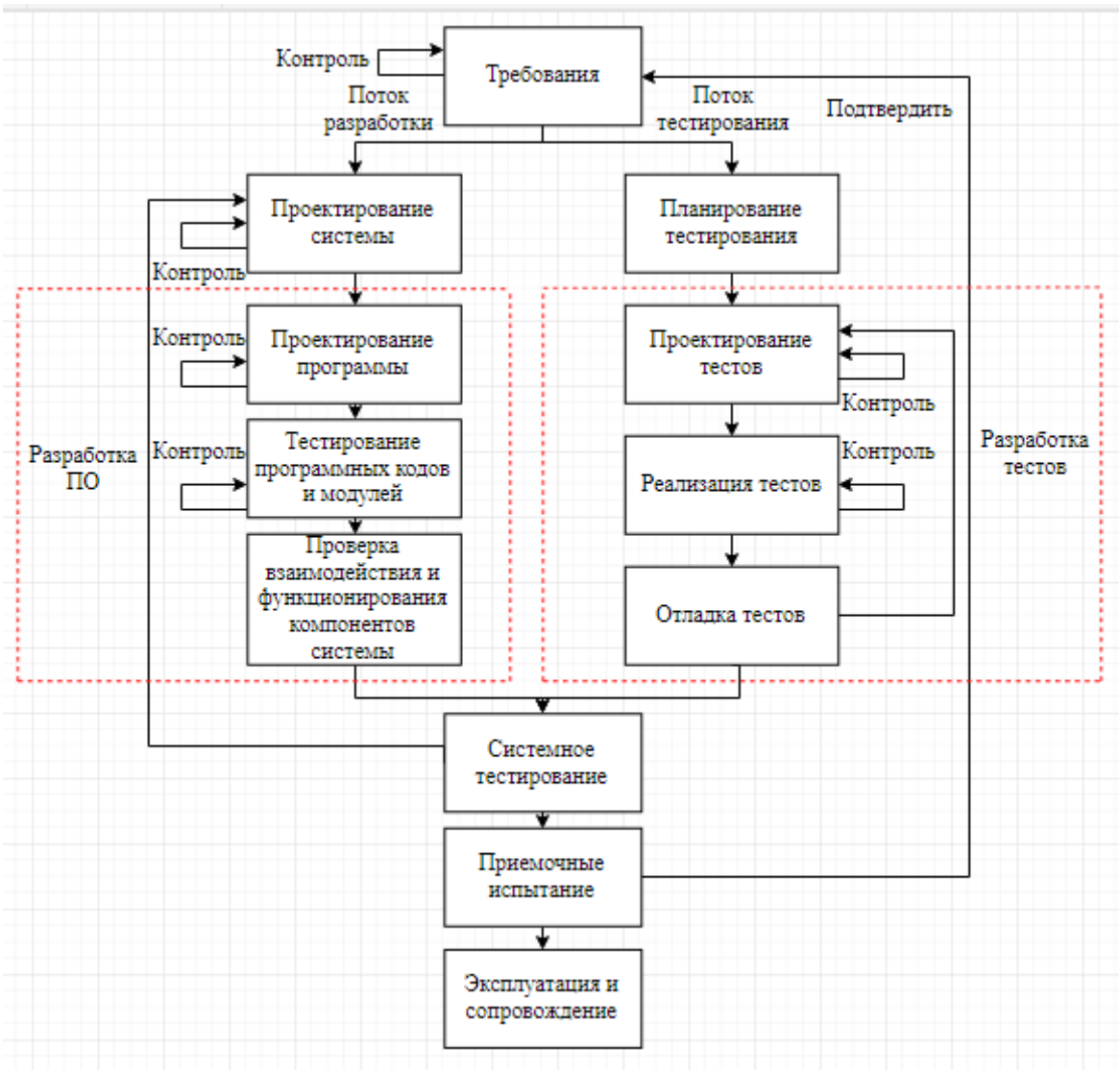


Рисунок 1.2 – Связь процессов тестирования и разработки ПО

Тестирование приложений на реальных устройствах позволяет предоставить клиентам по-настоящему качественные продукты и повысить эффективность бизнеса в сфере предоставления услуг.

Стоит заметить, что процесс тестирования проходит в рамках последовательности выполнения поставленных задачи, выбирается программа или приложение, которое будет проходить тестирование, на соответствие требованиям установленных при разработке и действующих стандартов качества. После проверки на критерии соответствия требованиям, происходит анализ и сбор информации, в результате чего будет получена информация о несоответствиях между ожидаемым результатом и фактическим.

Действующий процесс тестирования можно рассмотреть на уровне концептуальной модели, с целью определить протекание процесса и всех этапов прохождения тестирования. Для этого была построена специальная модель, AS-IS по-другому модель «КАК ЕСТЬ», это означает, что эта модель является уже существующей моделью процесса или функции. Исследование процессов является обязательной частью при проектировании или развитии системы. Модель «КАК ЕСТЬ» предназначена для точного фиксирования процессов, которые осуществляются в системе, а так же какие объекты используются при выполнении каких либо процессов или функций, различного уровня детализации.

Во время построения модели «КАК ЕСТЬ», очень важно спроектировать модель, максимально приближенную к действительности, которая основывается на естественно присутствующих процессах в системе.

В данном случае, продемонстрированный ниже процесс, рассматривается как процесс тестирования с помощью готовой системы тестирования, с уже имеющимися данными, написанными сценариями и готовыми тест-кейсами.

Контекстная диаграмма «КАК ЕСТЬ» процесса «Тестирования» представлена на рисунке 1.3.

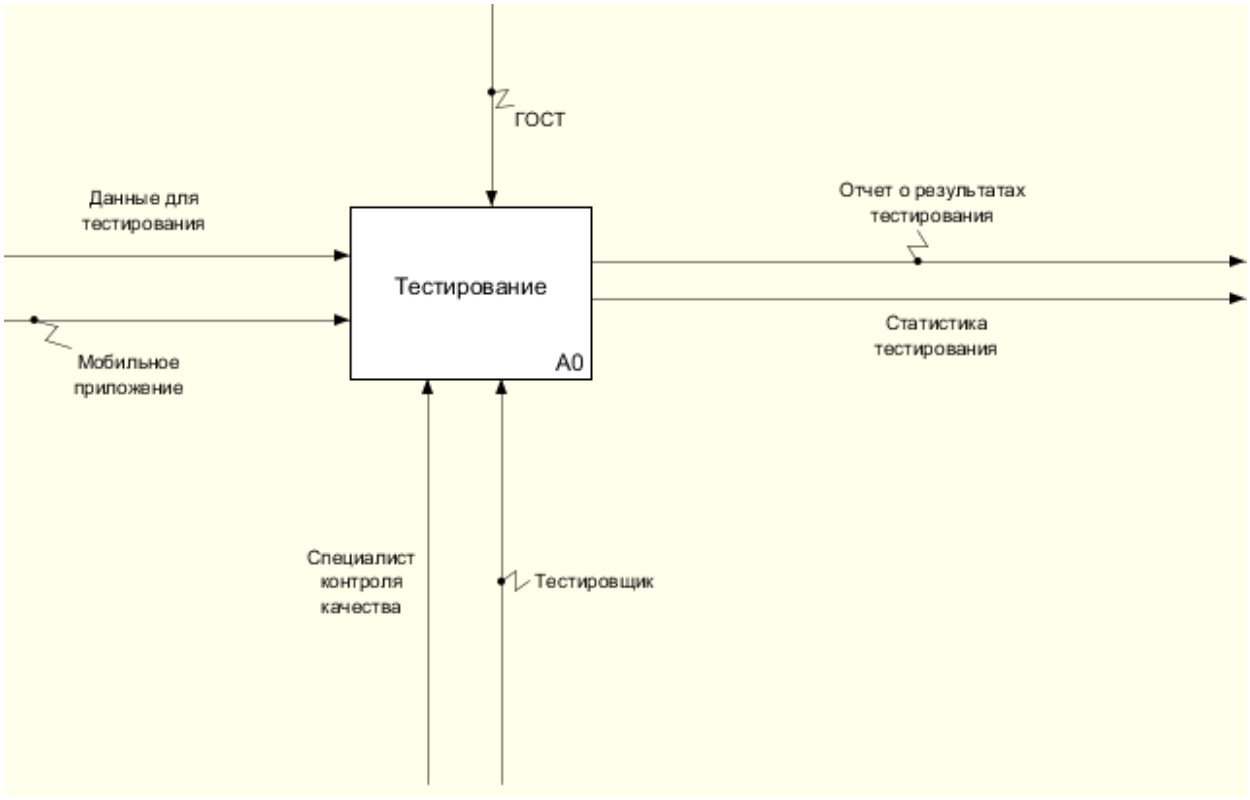


Рисунок 1.3 – Концептуальная модель тестирования

После того, как система будет описана в целом, проводится ее разбиение на фрагменты, такой процесс называется функциональной декомпозицией, а диаграммы, которые участвуют в описании каждого фрагмента и взаимодействия этих фрагментов, называются диаграммой декомпозиций. Диаграммы потоков данных «DataFlowDiagrams» или по-другому DFD - диаграмма, предназначена для демонстрации функциональных процессов, которые связываются потоками данных. С помощью такой диаграммы можно увидеть процесс преобразование входных данных в выходные и установить отношения между этими данными. Данная диаграмма так же используется для установления связи между протеканием процессов тестирования.

В данной диаграмме, продемонстрированный процесс, так же рассматривается как процесс тестирования с помощью готовой системы тестирования, с уже имеющимися данными, написанными сценариями и готовыми тест-кейсами.

На рисунке 1.4. представлена диаграмма декомпозиции основного процесса тестирования.

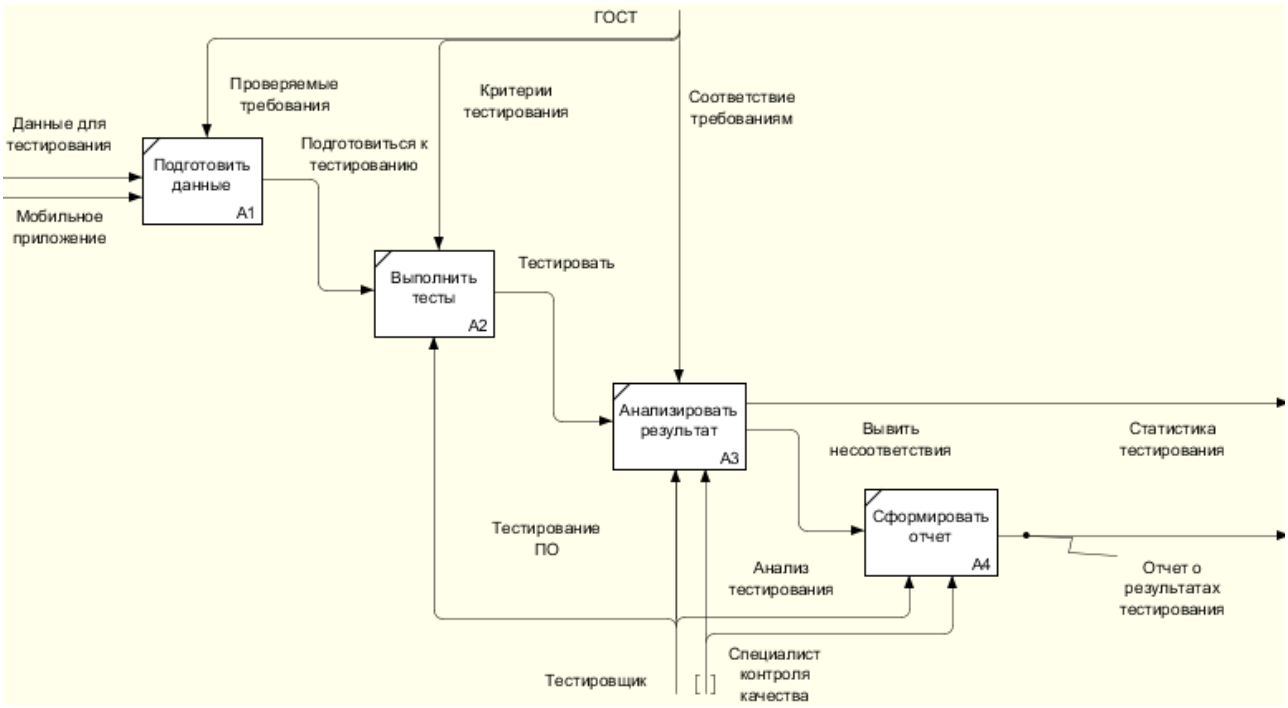


Рисунок 1.4 – Декомпозиция процесса тестирования «КАК ЕСТЬ»

На входе будет получена программа, которую необходимо тестировать и данные к тестированию. Наблюдая за программой в определенных условиях, на выходе будет получена информация о соответствии или несоответствии программы установленным требованиям. Данная информация будет использована для исправления ошибок в действующем продукте, любо для изменения требований к продукту еще на стадии разработки.

Тестирование включает в себя выбранную определенным образом, искусственно созданную ситуацию, по сымитированным запросам и описанием наблюдений, которые нужно осуществить, для проверки программы на соответствие определенным требованиям.

Тест может занять совсем немного времени, так и наоборот, например, тест производительности, проверяющий работоспособность системы при длительной нагрузке.

Таким образом, в процессе тестирования, человек выполняющий проведение тестирование продукта, выполняет две основные задачи:

1. Первой задачей является управление выполнением программы, а также создание искусственных ситуаций, в которых и происходит проверка поведения программы.

2. Вторая задача состоит в наблюдении за тем, как программа ведет себя в различных созданных ситуациях, и в сравнении того, что он видит с тем, что ожидается.

Тестированием является процесс выполнения проверки на предмет соответствия программного обеспечения требованиям, заявленным заказчиком. Оно осуществляется в специальных, искусственно сымитированных ситуациях посредством наблюдения за работой программного обеспечения. Именно такие, искусственно построенные ситуации называют процессом тестирования.

Тест-кейс является минимальным элементом тестирования, в котором интегрируются конкретные действия, а так же условия и параметры, задача которых осуществлять проверку определенной функциональности. Набор тест-кейсов называется тестовым набором (test suite).

Тест-кейсы позволяют при тестировании продукта выполнить проверку без полного ознакомления с документацией. При условии, что созданный тест-кейс удобен в поддержке, то, написанный один раз, он позволит сократить затрачиваемое время на тестирование в будущем и улучшить процесс тестирования. Подробные тест-кейсы так же способны значительно снизить вариативность выполнения тестов различными тестировщиками, что повышает качество тестирования программного продукта.

Рассматривая задачи современного тестирования, можно сделать вывод, что предназначены они не только для обнаружения ошибок в программах, но и в выявлении причин, по которым возникают данные ошибки. Такой подход к процессу тестирования позволяет разработчикам выполнять свою работу с максимальной эффективностью, устраняя обнаруженные ошибки быстро и своевременно.

Анализируя эту информацию, можно с точностью определить всю важность и необходимость выполнения тестирования.

1.3 Анализ видов и методов тестирования

В настоящее время существует достаточно большое число методов тестирования программного обеспечения.

При тестировании приложения, любой вид тестирования является необходимым. Важно понимать, с какой целью выполняется тестирование и что необходимо делать с полученным результатом. Если произвести тестирование и при этом результат просто проигнорировать, то совершенно не было смысла проводить тестирование, так как в таком случае это пустая трата времени и сил. Стоит отметить, что любое тестирование включает в себя:

1. Планировка теста, здесь необходимо исходить из целей тестирования.

2. Проектирование теста, здесь необходимо разработать сами тесты, то есть, скрипты, сценарии и т.д.

3. Воспроизведение теста, собственно выполнения задуманного, с помощью средств которые заранее были подготовлены.

4. Анализ результатов, здесь необходимо установить, что необходимо в дальнейшем делать с полученными результатами, и что необходимо сделать для исправления.

К наиболее важным и основным методами тестирования относятся:

1. функциональное тестирование;

2. тестирование производительности;

3. тестирование безопасности;

4. тестирование удобства пользования (юзабилити-тестирование);

5. тестирование совместимости;

6. тестирование на восстановление.

При этом стоит, отметь, что это далеко не все методы тестирования, есть так же и другие виды которые включают в себя достаточно большое количество этих методов.

Эти самые виды и методы, не смотря на то, что не были описаны ранее, являются не менее важными в любом процессе тестирования, виды и методы продемонстрированы ниже на рисунке 1.8

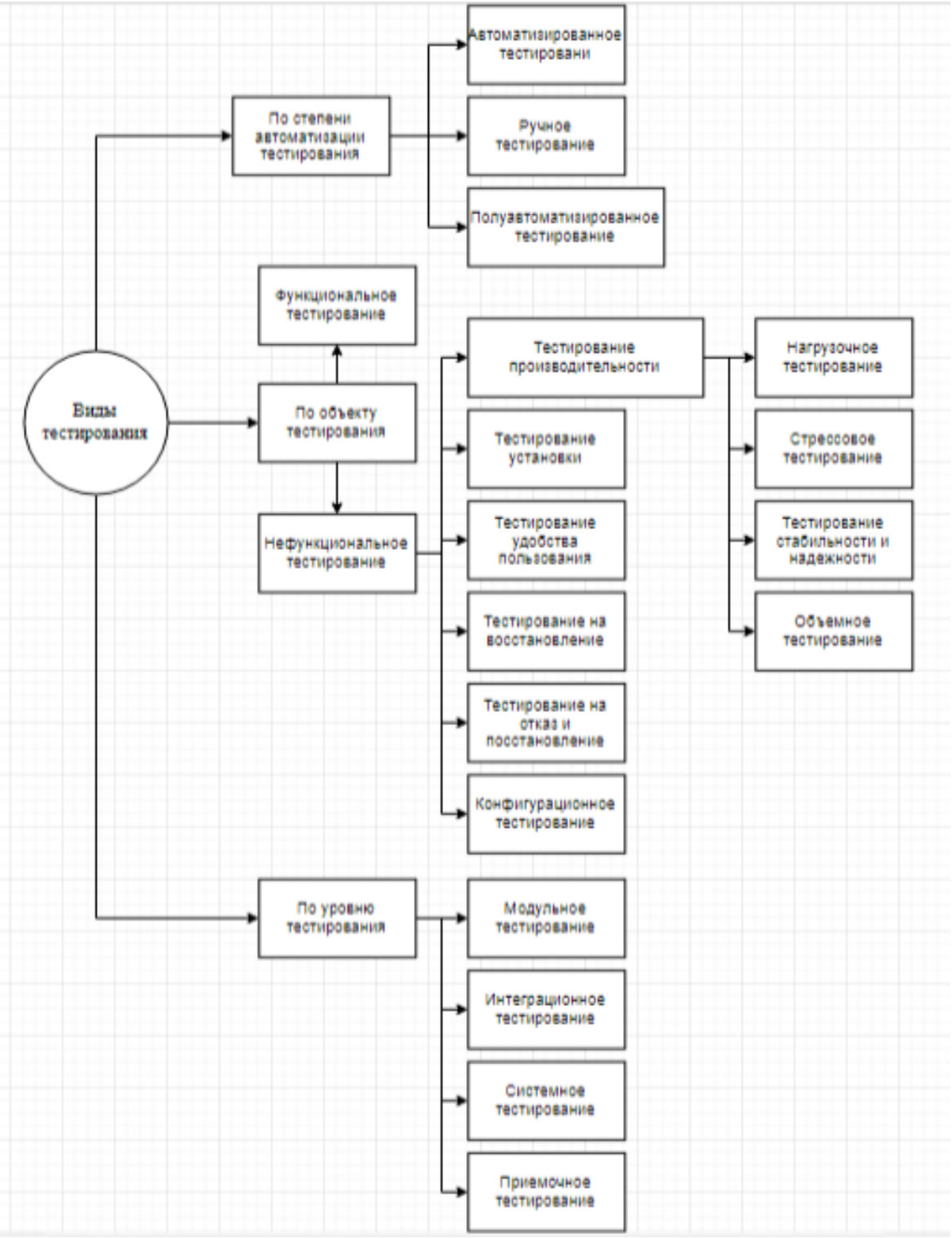


Рисунок 1.5 – Классификация видов и методов тестирования

Стоит подчеркнуть, что одной из наиболее развитых классификаций является по объекту тестирования. На рисунке так же указана данная классификация. Именно в эту классификацию входит метод нагрузочного

тестирования, который будет рассматриваться. Классификация и методы тестирования, которые в нее входят, указаны на рисунке 1.9

Рисунок 1.6 – Классификация по объекту тестирования

В этой связи можно выделить следующие виды тестирования, связанные с этим:

1. Функциональное тестирование.

2. Нефункциональное тестирование.

Функциональное тестирование в наши дни является наиболее приоритетным видом тестирования.

Задачей данного тестирования является установление соответствия программного обеспечения требованиям заказчика с точки зрения функциональности программного продукта.

Другими словами, проведение функциональных тестов позволяет проверить способность информационной системы решать задачи пользователей.

Нефункциональное тестирование позволяет оценить работу нефункциональных частей готового программного продукта.

Другими словами, нефункциональным тестированием является тестирование всех свойств программы, не относящихся к функциональности системы. В качестве таких свойств можно выделить следующие характеристики:

1. Производительность (способность системы работать на максимальных нагрузках).
2. Удобство (исследование удобства работы пользователя с приложением).
3. Масштабируемость (возможность масштабировать приложение как вертикально, так и горизонтально).
4. Безопасность (исследование ситуаций, при которых может быть нарушена работа приложения и кража персональных данных злоумышленниками).
5. Портируемость (возможность перенесения приложения на другие платформы).

Другими видами нефункционального тестирования также являются:

1. Тестирование пользовательского интерфейса.
2. Тестирование удобства использования.
3. Тестирование безопасности.
4. Установочное тестирование.
5. Конфигурационное тестирование.
6. Тестирование надежности и восстановления после сбоев
7. Тестирование локализации.
8. Нагрузочное тестирование.
9. Тестирование стабильности.
10. Объемное тестирование.
11. Тестирование переносимости.

На основании этих данных рассматривается комплексное тестирование и выявление необходимости проведения такого тестирования.

При выполнения работы основным условием заказчика, являлось уделить наибольшее внимание тестированию производительность. В тестировании производительности есть свои методологии тестирования такие как:

1. Нагрузочное (load).

2. Стресс (stress).

3. Тестирование стабильности (endurance or soak or stability).

4. Конфигурационное (configuration).

Так же возможно несколько подходов при тестировании производительности программного обеспечения:

1. В термине рабочей нагрузки: программное обеспечение подвергается тестированию по сценариям максимально приближенным к реальным условиям.

2. В рамках, бета-тестирования, когда приложение подвергается тестированию реальным пользователем.

1) Тестирование нагрузки представляется собой простую форму тестирования производительности. Такое тестирование обычно выполняется с целью оценки поведения приложения при различных вариациях и степенях нагрузки. Такой нагрузкой может быть: большое число пользователей, одновременно работающих с приложение, число каких либо транзакций, совершаемое за определённый интервал времени. С помощью такого типа тестирования чаще всего удаётся получить время отклика основных транзакций. Если же проводить наблюдение за базой данных, сервером приложения, сетью и т.п., то с помощью такого типа тестирования можно так же выявить уязвимые места в приложении.

2) Стрессовое тестирование, такой тип используется в основном, для установления пределов пропускной способности приложения. Он так же необходим для определения надёжности системы во время экстремальных или

диспропорциональных нагрузок, и показывает результат производительности в момент превышения текущий нагрузки ожидаемого максимума. 3)

3) Тестирование стабильности, такое тестирование позволит убедиться, что приложение во время длительной работы сможет выдержать определенную нагрузку. Во время тестирования, внимание уделяется потреблению ресурса памяти, тестируемым приложением, это позволит обнаружить потенциальные утечки. Так же с помощью такого типа тестирования можно выявить какую-либо деградацию производительности, которая будет выражаться в значительном снижении скорости обработки информации, а так же обнаружить в процессе длительной работы приложения, с момента начала тестирования, происходит ли увеличение времени ответа, после продолжительной работы по сравнению с началом теста.

4) Конфигурационное тестирование, представляет собой еще один тип тестирования производительности. Во время проведения такого тестирования, производительность оценивается не со стороны повышения нагрузки, а выявляется эффект влияния на изменения в конфигурации. Конфигурационное тестирование, можно совмещать с другими типами тестирования производительности: тестированием нагрузки, стрессоустойчивости и стабильности.

В тестировании производительности есть так же свои определенные принципы и экспериментальные факты, которые применяются при проведении данного типа тестирования. Если рассматривать в целом, то применяются они при тестировании производительности, к любому из типов тестирования, в частности, к тестированию нагрузки.

Далее, необходимо рассмотреть, что же представляют собой принципы и факты, которые применяются в тестирования производительности, поскольку они будут являться важными аспектами при проведении данного вида тестирования.

Принципы и экспериментальные факты тестирования производительности:

1. Уникальность запросов.

Когда формируются сценарии работы с тестируемой системой, основываясь на ее статистики, важно учесть факт того, что даже при самом реалистично сформулированном сценарии, могут обнаружиться исключения.

2. Время отклика системы.

Такой показатель находится в подчинении у функции нормального распределения. Под этим подразумевается, что при условии, достаточного количества измерений, появляется возможность определения вероятности, с которой отклик системы на запрос, будет попадать в какой-либо фиксированный интервал времени.

3. Зависимости между временем отклика системы и степенью распределённости системы.

Под этим подразумевается, что дисперсия нормального распределения отклика системы на какой-либо запрос, будет пропорциональная отношению числа узлов в системе, где запросы обрабатываются параллельно и количество запросов которые приходятся на каждый узел. Под этим подразумевается, что количество запросов, которые приходятся на каждый узел в системе, добавляют случайную величину задержки, при обработке запросов, оказывают влияние на разброс значений времени отклика.

4. Разброс времени отклика системы.

Исходя из вышеупомянутого утверждения, можно прийти к выводу, что при наличии большого числа измерений, значений времени обработки запроса в системе, всегда обнаружатся запросы, время обработки которых будет превышать установленный требованиями максимум, при этом стоит заметить, что новые максимумы будут выше, это будет обусловлено повышением суммарного времени проведения эксперимента. При формировании требований к производительности системы очень важно это учитывать, так же не менее важно учесть это, при регулярном выполнении тестирования нагрузки.

5. Точность воспроизведения профилей нагрузки.

Если в системе содержится большое количество компонентов, необходимая точность воспроизведения нагрузки будет обходиться дороже. Как бывает в большинстве случаев, нет возможности, учесть все критерии профиля нагрузки для сложных систем. Это обусловлено сложностью системы, ведь от этого будет зависеть количество времени, затраченное на проектирование, программирование и поддержку естественного профиля нагрузки для этой системы, стоит отметить, что это не всегда является необходимостью. В таком случае, оптимальный подход, будет обеспечиваться в балансе между стоимостью разработки теста и покрытием функциональности системы, вследствие чего, появиться возможность обнаружить допущения, которые в значительной степени будут влиять на общую производительность, в определенной части тестируемой системы.

1.4 Анализ целевого назначения методологий тестирования

У каждой методологии тестирования есть свои, четко поставленные назначения и основные задачи, которые должны выполняться во время тестирования.

1) Функциональное тестирование мобильных приложений, подразумевает собой тестирование взаимодействия с пользователем, а также тестирование транзакций. К наиболее важным факторам для этого тестирования относиться:

1. Тип приложения, определяется бизнес-функциональностью, этого приложения.

2. Аудитория, для которой предназначается приложение.

3. Каналы, по которым будет распространяться приложение.

В целом, функциональное тестирование необходимо для проверки выполняемых функций в приложении, которые описаны в спецификации или предусмотрены бизнес-процессами. Именно по этой причине, функциональное тестирование можно выполнять, основываясь на требованиях. Для таких случаев формируются тест-кейсы в основе которых заложено техническое

задание, основанное на бизнес-процессах. После, используются юз-кейсы, называемые сценариям, которые позволяют проверить приложение на основе постоянного использования.

Как часто бывает, система обладает большим количеством функций, и не всегда есть возможность проверить их все. В этом случае перед проведением функционального тестирования обычно выявляется приоритеты для тех или иных тест-кейсов и юз-кейсов, в соответствии с расставленными приоритетами распределяют время, после чего уделяют внимание более важным. Определить какие-либо стандартные сценарии для функциональных тестов сложно из-за разнообразия приложений, но можно выделить часто встречающиеся модули, составить для них тест-кейсы и в дальнейшем использовать их, модифицируя под конкретные требования.

2) Тестирование производительности, его можно определять как нагрузочное тестирование. Такое тестирование является автоматизированным, и имитирует работу определённого количества пользователей в какой-либо системе.

К основным задачам такого тестирования относятся:

1. Определение количеств пользователей с возможностью одновременной работы в приложении.

2. Произвести проверку поведения приложения при увеличении интенсивности выполнения каких-либо операций внутри системы. Проверить работоспособность приложения при длительном использовании при средней нагрузке.

3. Произвести проверку поведения приложения в стрессовых ситуациях.

4. Произвести проверку работы в условиях «расширенной» базы данных, определить скорость обработки запросов.

Главней задачей такого вида тестирования является определение приемлемой работоспособности приложения при определенных требованиях

производительности: наличие доступа для большого числа пользователей, устранение важно элемента инфраструктуры, к примеру, сервера базы данных.

К основным сценариям тестирования производительности мобильных приложений относятся:

1. Определить, одинаково ли работает приложение при разных условиях загрузки сети.

2. Определить, способно ли текущие покрытие сети обеспечить работу на разных уровнях пользовательской нагрузки.

3. Определить, может ли действующая клиент-серверная конфигурация обеспечить оптимальную производительность.

4. Выявить проблемные места в приложении и инфраструктуре, которые могут снизить производительность приложения.

5. Определить соответствие требованиям время реакции приложения.

6. Произвести оценку возможности приложения и аппаратного обеспечения справиться с планируемым объемом нагрузки.

7. Произвести оценку времени, в течение которого будет обеспечиваться бесперебойная работа приложения, к примру это так же может быть, время работы аккумулятор устройства при работе приложения с определенной нагрузкой.

8. Определить работу приложения в случае перехода из Wi-Fi-сети в мобильную сеть и наоборот.

9. Определить правильность работы уровней памяти процессора, а так же оптимальность самой работы.

10. Определить, что потребление батареи и утечка памяти не выходят за пределы нормы, а работа различных ресурсов и сервисов, таких как GPS-навигация или камера, соответствует требованиям.

11. Определить стойкость приложения в условиях большой нагрузки.

12. Определить эффективную работу сети при условии, что устройство находится в движении.

13. Определить производительность приложения, с учетом непостоянного подключения к сети Интернету.

3) Тестирование безопасности, такое тестирование необходимо для оценки безопасности приложения, кроме того, для произведения анализа рисков, при целостном подходе к защите приложения: возможная атака хакеров, ошибки и вирусы, несанкционированный доступ к конфиденциальным данным.

Главная задача такого тестирования, обеспечение безопасности сети и данных приложения.

Ключевыми действиями, для проверки безопасности приложения являются:

1. Убедиться в защите персональных данных пользователя, таких как: логин, пароль, номер карты и т.п., данные должны быть защищены от сетевых атак и не должны быть найдены путем подбора.

2. Убедиться, что приложение не предоставляет доступ к защищенному контенту или функциональности, без необходимой аутентификации пользователя в системе.

3. Убедиться в надежности вводимого пароль и исключить возможность взломщика завладеть паролем.

4. Убедиться, что таймаут для сессии во время аутентификации соответствует требованиям.

5. Выявить динамически зависимости в системе и обеспечить защиту от взлома.

6. Обеспечить защиту приложения от атак типа SQL-injection.

7. Выявить случае ошибочного кода, который не поддается управлению и принять меры по его исправлению.

8. Убедиться, что срок действия сертификатов не истек.

9. Обеспечить защиту приложения от DoS-аттак.

10. Произвести анализ требований к хранению и проверки данных.

4) Юзабилити-тестирование, необходимо для создания быстрых и простых в обращении приложений. Главной задачей является обеспечение приложения простым и удобным интерфейсом, то есть, создать интуитивный, соответствующий принятым стандартам интерфейс.

Для юзабили –тестирования ключевыми задачами является:

1. Убедиться, что кнопки в приложении имеют нормальный размер и подходят для всех категорий пользователей.

2. Помести кнопки в одной области, что бы не вызывать замешательство у пользователя во время работы.

3. Убедиться, что значки и картинки в приложении смотрятся естественно.

4. Убедиться, что кнопки, выполняющие одинаковые функции выглядят максимально приближенными друг к другу

5. Убедиться в наличии корректной работы функции уменьшения и увеличения.

6. Обеспечить минимальный ввод данных с клавиатуры.

7. Убедиться в возможности возврата или отмены при некорректных действиях.

8. Убедиться, что контекстуальное меню не перегружается. Так как предполагает быстрое использование.

9. Убедиться, что любой текст хорошо видим пользователю.

10. Убедиться в возможности прочесть короткие предложения и абзацы.

Юзабилити-тестирование, чаще всего проводится с помощью пользователей, поскольку, только пользователь сможет передать свои субъективные ощущения от использования приложения.

5) Конфигурационное тестирование, предназначено для возможности обеспечить оптимальную работу приложения на разных устройствах, с учетом их размера, разрешения экрана, версии , аппаратного обеспечения и т.п.

Главные задачи конфигурационного тестирования:

1. Убедиться, что приложение соответствует устройству, работает корректно.

2. Убедиться, что любой текст читается удобно, приложение подстраивается под разрешение экрана.

3. Убедиться, что некоторые функции, например вызова/будильника, доступны при запущенном приложении, что приложение будет сворачивать в случае входящего звонка, а по завершению возобновляться. Такой тип предназначен для проверки приложения в конфигурациях системы.

6) Тестирование на восстановление, предназначается для проверки приложения на возможность возобновления работы после непредвиденных сбоев, возникших в связи с ошибками в работе, отказом оборудования или проблемами связи. Применяется чаще всего в приложениях для категории 24x7, где очень важна каждая минута.

Для такого тестирования ключевыми задачами является следующее:

1. Проверять восстановление после сбоя в системе.

2. Проверять эффективность восстановления после непредвиденных ситуаций в сценарии.

3. Проверка эффективности работы приложения в случае сбоя сети или отключении питания.

4. Проверять восстановление данных в случае потери сигнала сети Интернета.

Другие важные области проверки:

1. Проверка на наличие нефункциональных клавиш.

2. Проверка экрана загрузки приложения.

3. Проверка возможности ввода данных при сбое в работе сети Интернет.

4. Проверка методов запуска приложения.

5. Проверка наличия эффекта зарядки в случае, если приложение находится в фоновом режиме.

6. Проверка функционирования экономичного режима и режима высокой производительности.

7. Выявление последствий извлечения аккумулятора во время работы приложения.

8. Проверка уровня потребления энергии приложением.

9. Проверка побочных эффектов приложения.

Исходя из анализа целевого назначения, задачи, которые предусматриваются в различных методологиях, тесно связанны между собой, они являются необходимыми для выявления несоответствий в работе тестируемого приложения и для обеспечения его качественной работы на должном уровне. В этом случае, можно подчеркнуть, что для достижения таких результатов проводится комплексное тестирование.

1.5 Анализ требований к проведению тестирования

Для выполнения тестирования необходимо соблюдать основные условия.

1. Естественным условием является тот факт, что невозможно выполнить тестирование по отсутствующему объекту. Основываясь на этом факте, выявляем первое условие: необходимо наличие доступного объекта тестирования, в рамках проведения испытаний.

2. Для того, что бы тестирование производилось нужно определить исполнителя, в этом случая можно заметить, что вторым обязательным условием, нужно наличие исполнителя. Стоит заметить, что в роли исполнителя будет выступать либо человек, либо машина или потребуется для проведения тестирования скомбинировать человека и машину вместе.

Так же необходимо определить еще некоторые условия, их можно соотнести к достаточным условиям:

1. Если учитывать тот факт, что не каждое совершенное в программе действие, которое позволит увидеть поведение, при тех или иных действиях, является тестированием, следует определить целевое назначение, в данном случае: тестирование, что и будет являться одним из достаточных условий.

2. Стоит принять во внимание, что перед проведением тестирования формируется план, отсюда можно выделить, что еще одним достаточным условием будет являться наличие плана тестирования.

3. Во время выполнения тестирования, будет появляться необходимость выполнять какие то действия для установления поведения программы между ожидаемым результатом и фактическим. Отслеживать данную процедуру будет человек либо машина, но для тестирования необходимы тестовые сценарии. В этом случае, еще одним достаточным условием, будет являться наличие тест-кейсов.

4. В качестве подтверждения проведения тестирования, потребуется отчет с результатами. В этом случае, необходимость отчета с подтверждением выполнения тестирования и результатами, так же будет являться достаточным условием.

В результате получаются следующие необходимые и достаточные условия для выполнения тестирования:

К необходимым условиям относятся:

1. Объект, доступный для проведения тестирования.

2. Исполнитель, который будет выполнять тестирование, и отслеживать результаты (человек или машина, либо комбинация человек и машина).

К достаточным условиям относятся:

1. Объект, доступный для проведения тестирования.

2. Исполнитель, который будет выполнять тестирование, и отслеживать результаты (человек или машина, либо комбинация человек и машина).

3. План выполнения тестирования.

4. Наличие тест-кейсов.

5. Отчет с результатами тестирования, который подтверждает выполнение ряда поставленных задач и целей.

Учитывая все вышеупомянутые условия для проведения тестирования, можно определить, что эти условия являются верными поскольку:

1. Формирование плана тестирования, говорит об активностях в области планирования.

2. Формирование каких-либо сценариев или тест кейсов, подтверждает наличие проектирования.

3. Выполнение тестирования свидетельствует о наличии исполнителя.

4. Формирование отчета по тестированию, говорит об анализе полученных результатов.

Так же стоит заметить, что тестирование проводится в рамках существующих общепринятых стандартов качества:

1. IEEE 829-2008 IEEE Standard for Software and System Test Documentation.

2. ANSI/IEEE Std 1008-1987 — IEEE Standard for Software Unit Testing.

3. ISO/IEC/IEEE 29119-1:2013 Software and systems engineering — Software testing — Part 1: Concepts and definitions.

4. ISO/IEC/IEEE 29119-2:2013 Software and systems engineering — Software testing — Part 2: Test processes.

5. ISO/IEC/IEEE 29119-3:2013 Software and systems engineering — Software testing — Part 3: Test documentation.

В результате стоит заметить, что перед проведением тестирования, необходимо учитывать и соблюдать вышеупомянутые требования, поскольку они являются важными аспектами для качественного и продуктивного выполнения тестирования.

1.6 Анализ архитектуры мобильных приложений

На сегодняшний день мобильные приложения реализуются в архитектуре, которая носит название «клиент-сервер ». Для клиент-серверной архитектуры предусматривается несколько видов:

1) Двухзвенная «клиент-серверная » архитектура

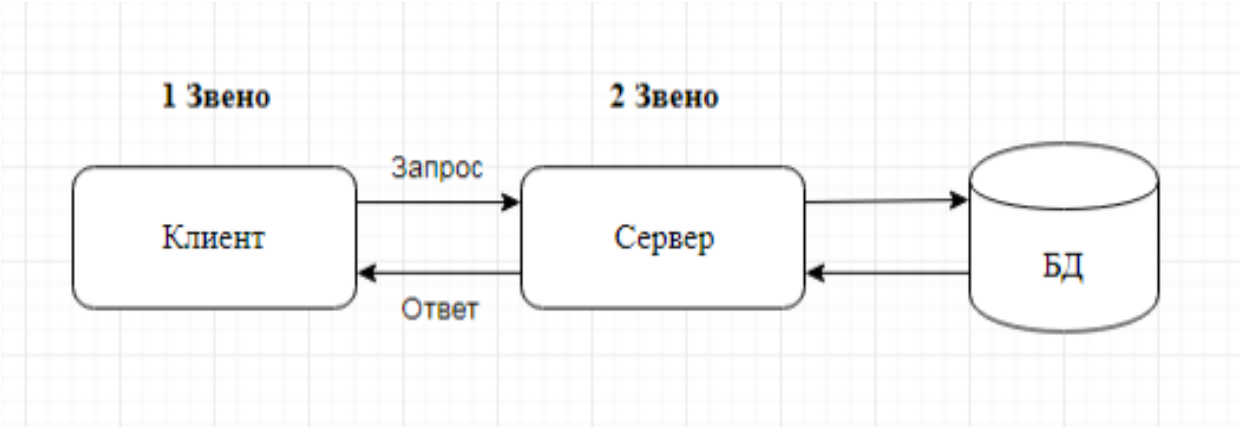


Рисунок 1.7 – Двухзвенная «клиент-серверная» архитектура

Первым звеном этой схемы, будет являться устройство пользователя с установленным на нем приложением, благодаря которому будет обеспечиваться взаимодействие пользователя с БД через сеть.

Вторым звеном будет являться, сервер баз данных, который принимает участие в обработке данных.

Использование двухзвенной архитектуры, способствует снижению нагрузки на информационную сеть, посредством передачи запроса и ответа.

Для такой архитектуры можно выделить следующие плюсы и минусы.

1) Плюсы:

1. Простота в реализации.

2. Достаточно низкая стоимость серверного оборудования.

2) Минусы:

1. Использование устаревших IT-решений.

2) Трехзвенная «клиент-серверная » архитектура

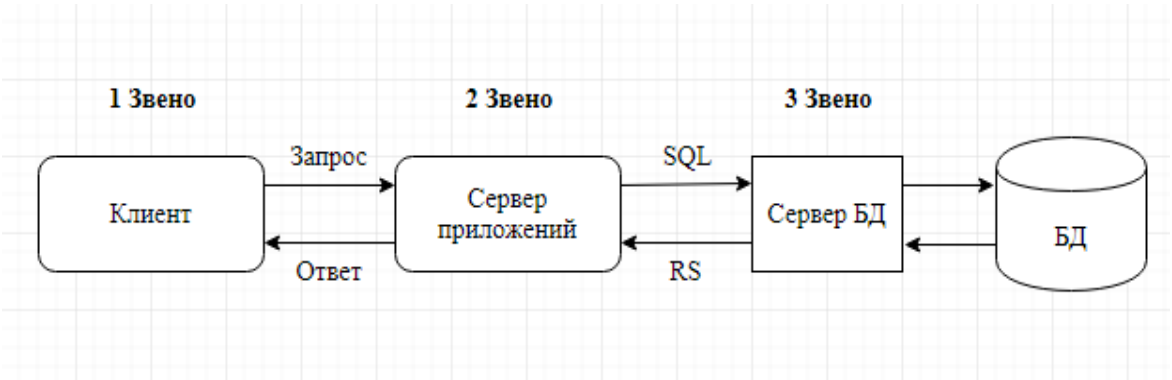


Рисунок 1.8 – Трехзвенная «клиент-серверная» архитектура

Первым звеном этой схемы, будет являться мобильно приложение на платформе Android.

Вторым звеном, сервер приложений, который является программным продуктом переходного уровня. Его назначение производить синхронизацию в работе компонентов системы и обеспечивать их взаимосвязь. Чаще всего, такую задачу выполняют сетевые службы и программные обеспечения.

Третьим звеном, является сервер баз данных, который не взаимодействует прямым образом с клиентом, что повышает безопасность системы.

Для этой архитектуры можно так же выделить следующие плюсы и минусы.

1) Плюсы:

1. Наличие масштабируемости.

2. Легкость конфигурирования.

2) Минусы:

1. Затруднение в воспроизведении программного продукта.

2. Достаточно высокие требования к продуктивности сервера баз данных и серверов приложений.

Для этих архитектур так же проводился сравнительный анализ, с целью выявления наиболее подходящей структуры при реализация программного продукта.

Сравнительный анализ этих архитектур, продемонстрирован в таблице 1.3.

Таблица 1.3 – Сравнительный анализ архитектур

Характеристика	**Двухзвенная модель «Клиент-сервер»**	**Трехзвенная модель «Клиент-сервер»**
Конфигурируемость	-	+

Характеристика	Двухзвенная модель «Клиент-сервер»	Трехзвенная модель «Клиент-сервер»
Масштабируемость	-	+
Низкие затраты на разработку и использование	+	-
Минимальные требования к оборудованию	+	-
Поддержка современных технологий	-	+
Итого:	2	3

По результатам такого анализ, для реализации мобильного приложения, более оптимально использовать трехзвенную архитектуру «клиент-сервер»

Вывод по первой главе

В первой главе, рассматривался процесс тестирования в целом: необходимость его выполнения, задачи которые выполняются в процессе тестирования, условия, которые необходимы для выполнения тестирования. Так же рассматривались виды и процессы тестирования, составляющие атрибуты любого тестирования.

По результатам выполнения анализа предметной области было произведено сравнение видов тестирования, а так же методологий.

По результатам анализа: при проектировании и разработке системы тестирования мобильных приложений, будет предусматриваться реализация следующих методов тестирования, однако, наибольший приоритет будет направлен в сторону тестирования производительности в частности нагрузочного тестирования:

1. Тестирование производительности.
2. Тестирование удобства пользования.

3. Конфигурационное тестирование.

4. Тестирование на восстановление.

Удалось проанализировать назначения тестирования, этапы выполнения тестирования, выяснить целевое назначение различных видов и методологий тестирования мобильных приложений.

Определить необходимость в проведении комплексного тестирования мобильного приложения.

Выявить необходимые условия, для выполнения тестирования, которые нужно соблюдать для достижения результативности.

Производился анализ существующих систем для проведения тестирования, которые не сегодняшний день так же являются эффективными и выполняют поставленные задачи, произвести оценку качеств этих систем, а так же определить положительные и отрицательные качества той или иной системы.

Проводился анализ существующих архитектур для мобильных приложений, выявлялись положительный и отрицательные стороны, той или иной архитектуры, с целью определения наиболее оптимальной.

ГЛАВА 2 ВЫБОР СРЕДСТВ РАЗРАБОТКА ДЛЯ СИСТЕМЫ ТЕСТИРОВАНИЯ МОБИЛЬНЫХ ПРИЛОЖЕНИЙ

2.1 Особенности разработки и анализ выбора средства

В последние годы наблюдается высокая актуальность высокотехнологичных устройств связи. Это порождает необходимость разработки для них различных приложений. Качество и производительность приложений в большей степени зависит от аппаратных характеристик. Однако удобство пользования в большей степени зависит от мобильной операционной системы.

Для дизайна мобильных приложений характерна многоплановая и многоуровневая организация.

В этом случае необходимо обратить внимание на визуальную иерархию.

Необходимо расположить элементы управления так, чтобы самые важные из них выделялись сильнее, находясь при этом на одном уровне с другими элементами.

В дальнейшем, так же следует учитывать, линию взгляда пользователя. Как принято считать, большинство пользователей просматривает экран слева-направо и сверху-вниз. Если учитывать такой фактор, стоит более тщательно разработать сценарий использования приложений для того, чтобы пользователь мог интуитивно найти и использовать элементы управления которые ему будут необходимы.

Одним из важных свойств дизайна интерфейса является гибкость. Под этим подразумевается следующее: мобильное приложение должно качественно и гармонично выглядеть на любом устройстве. Ведь при большом количестве различных размерах экранов, достаточно сложно сохранить один и тот же формат экрана. Чтобы не сохранять его, а производить изменения, в зависимости от характеристики используемого устройства, необходимо использовать метод модульного программирования в процессе разработки.

Суть этого метода заключается в том, что при непосредственном программировании дизайна, большая часть наиболее значительных пунктов

разбивалась на отдельные самостоятельные части, в дальнейшем именуемые модулями. Ведь в таком случае, для внесения изменений в какую либо часть интерфейса, не проходилось исправлять весь код, потребуется внести изменения только в модуль, который и отвечает за тот или иной элемент, который требуется изменить.

Также стоит соблюдать единство дизайна при переносе приложения с одной платформы на другую. Это нужно, чтобы пользователь не терял способность к быстрой адаптации при переходе, например, с мобильной версии приложения к ее интернет-версии.

Важным этапом при разработке дизайна также является масштабирование элементов управления, ведь от того, как их расположить, во многом зависит степень удобства при использовании приложения.

Все перечисленные особенности организации приложений на мобильных устройствах позволяют сделать вывод о том, что использование браузеров при работе с системами дистанционного обучения является далеко не оптимальным решением. Так же необходимо помнить, что при реализации программного продукта, очень важно выбрать качественную операционную систему. Которая будет доступна и понятна при разработке приложения. Желательно выбирать операционную систему, с которой уже приходилось работать. В таблице приведен список достаточно популярных ОС для разработки приложений.

Таблица 2.1– Анализ популярных средств разработки

ОС	**Android**	**BlackBerry**	**iOS SDK**	**Python**
Язык программирования	Java, частично C, C++, Delphi	Java	Objective-C, Swift	Python

ОС	Android	BlackBerry	iOS SDK	Python
Доступный отладчик	Встроенные отладчики Android Studio и Eclipse, доступна автономная отладка	Встроенный в IDE отладчик	Встроенный в Xcode IDE отладчик	Да
Доступный эмулятор	Да	Да	в комплекте с iPhone SDK, встроенный Xcode IDE	Add-on для Nokia Emulator
Доступная IDE	Android Studio, Eclipse, Проект Kenai — плагин Android для NetBeans, IntelliJ IDEA	Eclipse	Xcode	Различные, включая плагины для Eclipse
Платформы для разработчиков	Android	BlackBerry	iPhone, iPad, i Pod Touch	Язык программирования доступен только на Nokia Series60, но существуют порты на другие платформы, включая PalmOS

ОС	Android	BlackBerry	iOS SDK	Python
Установочный пакет	apk	alx, cod	Только App Store, требует проверки и утверждения Apple Inc.	Sis внедрение с py2sis или можно использовать Python Runtime
Стоимость инструментов для разработки	Бесплатно, Delphi — доступна коммерческая лицензия	Бесплатно	Бесплатные инструменты для основанных на IntelMac. На симуляторе тестирование бесплатно, но для установки потребуется платный ключ от разработчика.	Свободный доступ

В качестве среды разработки была выбрана ОС Android Studio. В Android Studio, имеется возможность производить запуск приложений, используя эмулятор, либо на реальном устройстве. Инсталляция этой среды разработки достаточно проста, сама среда является доступной в понимании и идеально подходит для новичков в программировании, которые используют язык Java. Android Studio обладает следующим рядом качеств: удобный и легко-воспринимаемый интерфейс, авто-загрузка компонентов необходимых для разработки программного продукта и большой объем справочной информации.

2.2 Общие характеристики операционной системы Android

Платформа Android производит адаптации программного обеспечения общего назначения для мобильных устройств.

Такая платформа, обладающая большим функционалом, представляет собой программный тип данных, который по-другому называется стек, ОС с Linux в качестве основы, назначение которого управлять устройствами, памятью и процессами. Android, содержит в своей библиотеке функции связанные с мобильными телефонами, видео, графикой, программированием пользовательских интерфейсов и другими возможностями для мобильных устройств. Так же, в Android осуществляется поддержка множества функций платформы Java SE (Java Standard Edition), кроме того Android применяет свой собственные более функциональный и современный Фреймворк пользовательского интерфейса.

Существует так же виртуальная машина (Java Virtual Machine), которая предназначена для интерпретации исполняемого байт-кода Java. Обычно, виртуальная машина обеспечивает оптимизацию, чтобы помочь достичь высокой производительности, сравнимой с аналогичными уровнями транслируемых языков, например C и C++.

Кроме того, Android предлагает свой собственный вариант оптимизированного JVM с возможностью выполнять скомпилированные файлы классов Java. Это необходимо для преодоления ограничений, которые свойственны, для мобильных устройств, в большинстве случаев связанных с памятью, скоростью работы процессора и мощностью. Такая виртуальная машина именуется Dalvik VM. Простота и доступность в понимании языка Java, обширность библиотеки классов Android, превращает Android в очень удобную, функционально оснащенную и понятную платформу для написания программ.

Android Studio, совместима с платформой «Google App Engine» для быстрой интеграции в облаке новых API и функций. Эта среда разработки включается в себя разные API, например «Google Play», «Android Pay» и «Health». Начиная с версии 1.6. в среде присутствует поддержка всех платформ. Так же есть варианты среды Android, которые в значительной степени отличаются от версии Google Android, «Amazon Fire OS», является наиболее

популярной . В Android Studio можно создавать APK для этой операционной системы, поддержка Android Studio ограничивается онлайн-форумами.

Рисунок 2.1 – Схематичное представление программного стека Android

Android использует комплексный и направленный на результат подход к реализации мобильной платформы, в этом случае обычных решений которые основываются на JVM будет недостаточно. Android включает в себя все необходимое, что понадобится в реализации программного продукта: операционная система, драйвера устройств, библиотеки ядра, собственный интерфейс Java, оптимизированная версия Dalvik VM и среда разработки Java, все это содержится в одном пакете. В процессе разработки можно быть уверенным, что при создании приложения все нужные и важные библиотеке, будут находиться на мобильном устройстве. Именно такой, комплексный подход отличается от других вариантов решений, используемых при разработке в программировании для мобильных устройств.

Так же предоставлена информация по системным требованиям, которым должно соответствовать оборудование разработчика для нормального

функционирования. Информация по минимальным системным требованиям указана в таблице 2.2

Таблица 2.2 – Системные требования к ОС Android Studio

	Windows	**OS X**	**Linux**
Версия ОС	Microsoft Windows 10/8/7/Vista/2003 (32 или 64-bit)	Mac® OS X® 10.8.5 или выше, до 10.9 (Mavericks)	GNOME или KDE
Оперативная память	3 ГБ (минимум), 8 ГБ (рекомендуется); +1 ГБ для Android Emulator		
Свободное место на диске	2 ГБ минимум (500 МБ для IDE + 1.5 ГБ для Android SDK и образа системы эмулятора), 4 ГБ SSD рекомендуемое		
Версия JDK	Java Development Kit 8		
Разрешение экрана	1280 x 800 (минимум)		
Дополнительно	-	Java Runtime Environment (JRE) 6	GNU C Library (glibc) 2.15 или выше

Как видно из таблицы, системные требования достаточно низкие для современных персональных компьютеров, а значит и программа должна стабильно функционировать, без сбоев и задержек в работе.

2.3 Выбор языка программирования для разработки приложения

При проектировании и последующей разработке программного продукта необходимо учитывать важные критерии, такие как:

1. Наличие и поддержание технологии быстрой разработки приложения, базируемой на объектно-ориентированном программировании.

2. Наличие возможности поддержки трехзвенной архитектуры «клиент-сервер».

3. Наличие знаний языка разработчиком программного продукта.

4. Возможность поддержки системы управления базами данных.

Поскольку уже была выбрана ОС Android Studio, то и язык программирования будет Java, однако стоит провести анализ с целью выявления возможностей использования этого языка. На текущий момент, при создании мобильных приложений используют следующие, широко распространённые языки программирования: Java и Objective-C.

Java, является мульти-платформенным языком, объектно-ориентированного программирования. Используется в разработках приложений, на платформах типа Android. К недостатком относится, допускаемая потеря скорости при выполнении приложений. Однако, язык является достаточно простым и доступным в понимании, что несомненно является плюсом.

Objective-C, является компилируемым языком объектно-ориентированного программирования, используется в разработках корпорации Apple , будучи построенным на основах языка СИ и парадигм Smalltalk.

Сравнительный анализ этих языков программирования приведен в таблице 2.3

Таблица 2.3 – Сравнительный анализ языков программирования

Характеристики	**Java**	**Objective-C**
Использование для разработки на платформе Android	+	+
Объектно-ориентрованный подход к разработке программных продуктов	+	-
Поддержка трехзвенной архитектуры «клиент-сервер»	+	+
Наличие знаний языка разработчиком	+	-
Поддержка СУБД	+	+
Итого:	5	3

По результатам сравнительного анализа можно сделать выбор в пользу языка Java, который будете являться основным средством в разработке мобильного приложения.

2.4 Разработка диаграммы вариантов использования мобильного приложения

Рассмотрим проектируемую систему тестирования мобильных приложений с точки зрения диаграмм вариантов использования. Благодаря такой диаграмме появляется возможность обозначить внешние системы которые будут взаимодействовать с системой, а так же главные процессы и их взаимосвязь. Диаграммы вариантов использования позволят определить функциональную структуру системы, не уделяя внимания ее реализации. Так же предварительно определяются и классифицируются системные объекты. По результатам сформировавшейся модели, реализуется план для разработки системы. Для предметной области, необходима назначить актера, им будет являться: тестировщик. Так же необходимо определить возможности которые обязано иметь разрабатываемое приложение: тестировщик применяет систему для осуществления тестов и получения результатов тестирования. Приложение будет производить тестирования и предоставлять результат, посредством доступ к сети Интернет. Используя эти данные, выявляются прецеденты, которые необходимо реализовать в приложении.

Таблица 2.4 – Краткое описание прецедентов

Прецедент	**Краткое Описание**
1	**2**
Запуск приложения	Запуск мобильного приложения
Настройка тестов	Произвести настройку с указанием числа запусков и задержками
Задать тесты	Зафиксировать заданные в настройках тестов параметры
Указать web-ссылку	Указать web-ссылку в текстовом поле
Произвести запуск	Произвести запуск теста
Результат	Переход к результатам теста

Для основных прецедентов, системы тестирования была разработана схематическая диаграмма вариантов использования на рисунке 2.2, где актер, изображенный на рисунке, является тестировщиком, который будет взаимодействовать с системой, а прецедентами являются варианты использования или сервисные запросы, которые обеспечиваются системой. На диаграмме линиями показаны отношения ассоциаций, которые олицетворяют способность применения актером прецедента.

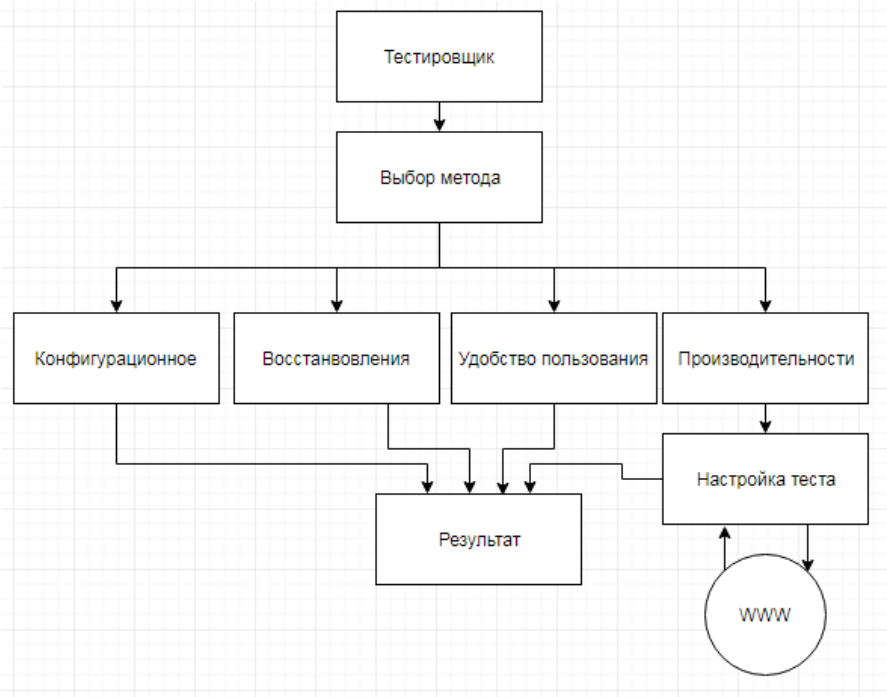


Рисунок 2.2 – Диаграмма вариантов использования

На рисунке 2.2 продемонстрированы действия, которые должен осуществить тестировщик для выполнения тестирования:

1. Осуществляет запуск приложения.
2. Производит настройку тестов.
3. Задает необходимые тесты.
4. Указывает web-ссылку (при необходимости).
5. Производит запуск системы тестирования.
6. Осуществляет анализ результатов.

Представим описание спецификаций в табличном виде для главных прецедентов. Демонстрация спецификаций главных прецедентов содержится в таблицах 2.5–2.10.

Таблица 2.5 – Описание прецедента «Запуск мобильного приложения»

Прецедент: Запуск мобильного приложения
ID:1
Краткое описание: Запуск мобильного приложения
Главные актеры: тестировщик
Второстепенные актеры: Нет
Предусловия: 1. Прецедент начинается по инициативе тестировщика
Основной поток: 1. Тестировщик запускает приложение
Постусловие: 1. Тестировщик получает доступ в главное меню приложения

Таблица 2.6 – Описание прецедента «Настройка Тестов»

Прецедент: Настройка Тестов
ID:2
Краткое описание: Произвести настройку с указанием числа запусков и задержками
Главные актеры: тестировщик
Второстепенные актеры: Нет
Предусловия: 1. Прецедент начинается по инициативе тестировщика
Основной поток: 1. Тестировщик нажимает кнопку "Настройки тестов". 2. Производит ввод необходимых данных. 3. Проверяет корректность, переходит к следующему шагу.
Постусловие: 1. Данные введены корректно

Таблица 2.7 – Описание прецедента «Задать тесты»

Прецедент: Задать тесты
ID:3
Краткое описание: Зафиксировать заданные в настройках тестов параметры
Главные актеры: тестировщик
Второстепенные актеры: Нет
Предусловия: 1. Прецедент начинается по инициативе тестировщика
Основной поток: 1. Тестировщик нажимает кнопку "Задать". 2. Происходит сохранение вводных данных.
Постусловие: 1. Возврат в главное меню

Таблица 2.8 – Описание прецедента «Указать web-ссылку»

Прецедент: Указать web-ссылку
ID:4
Краткое описание: Указать web-ссылку в текстовом поле
Главные актеры: тестировщик
Второстепенные актеры: Нет
Предусловия: 1. Прецедент начинается по инициативе тестировщика
Основной поток: 1. Тестировщик вводит название web-ссылки в поле "App". 2. Проверяет корректность введенных данных, переходит к следующему шагу.
Постусловие: 1. Данные введены корректно

Таблица 2.9 – Описание прецедента «Произвести запуск»

Прецедент: Произвести запуск
ID:5
Краткое описание: Произвести запуск теста
Главные актеры: тестировщик
Второстепенные актеры: Нет
Предусловия: 1. Прецедент начинается по инициативе тестировщика
Основной поток: 1. Тестировщик нажимает кнопку "Запуск" в главном меню. 2. Производится тестирование и сбор информации.
Постусловие: 1. Тестировщик ожидает окончания теста

Таблица 2.10 – Описание прецедента «Результат»

Прецедент: Результат
ID:6
Краткое описание: Переход к результатам теста
Главные актеры: тестировщик
Второстепенные актеры: Нет
Предусловия: 1. Прецедент начинается по инициативе тестировщика
Основной поток: 1. Тестировщик нажимает кнопку "Результаты" в главном меню. 2. Производится переход в окно результатов тестирования.
Постусловие: 1. Нет

В результате данного этапа были выявлены главные роли и выполняемые функции, которые способствуют функционированию приложения с выполнением тестирования.

Так же для данной системы предусмотрена диаграмма размещения компонентов мобильного приложения. Доступ к различным web-ссылкам, с мобильного приложения осуществляется через Интернет. Диаграмма компонентов предназначается для возможности формирования архитектуры разрабатываемой системы, где обозначаются программные компоненты и их соотношения. В диаграмме основными графическими элементами будут являть: компоненты, а так же интерфейсы с обозначенными между ними зависимостями. Сформированная диаграмма компонентов продемонстрирована на рисунке 2.3. По этой диаграмме видно взаимоотношение компонентов, а так же результаты их взаимодействия.

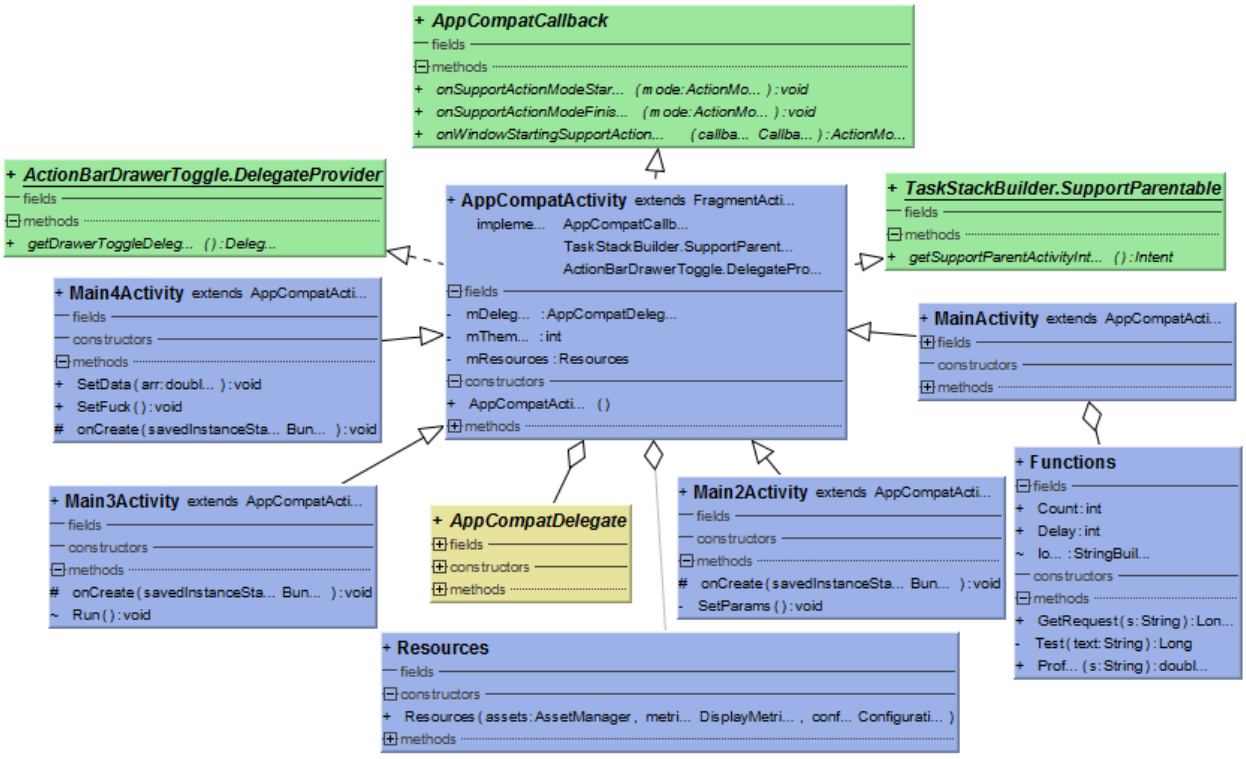


Рисунок 2.3 – UML Диаграмма компонентов

Сформированная диаграмма компонентов, демонстрирует наглядно, модули аппаратного и программных уровней, а так же их взаимосвязь.

1. Через главное меню «MainActivity» с помощью кнопки «настройки тестов», совершается переход к модулю настроек «MainActivity2».

2. После ввода параметров настройки, с помощью кнопки «задать», происходит фиксация заданных параметров и возвращение в главное меню «Main».

3. Далее в окне ввода указывается необходимая web-ссылка, после чего кнопкой «Запуск» производится запрос на сайт, с выставленными параметрами, по которым производится тест и сбор информации.

4. После выполнения теста происходит передача обратной информации.

5. С помощью кнопки «Результаты», совершается переход в модуль результатов «MainActivity4».

Вывод по второй главе

В данной главе был осуществлен анализ особенностей разработки мобильных приложений.

Анализ существующих средств разработки и языка программирования, с последующим выбором ОС Android Studio для реализации приложения, а так же с выбором язык программирования Java.

Проводился анализ характеристик операционной системы Android Studio.

Разрабатывалась диаграмма вариантов использования системы тестирования мобильных приложений, проводилось описание основных прецедентов.

ГЛАВА 3 ПРОЕКТИРОВАНИЕ И РЕАЛИЗАЦИЯ СИСТЕМЫ ТЕСТИРОВАНИЯ МОБИЛЬНЫХ ПРИЛОЖЕНИЙ

3.1 Проектирование алгоритм выполнения и реализация тест-кейсов

Для демонстрации работы был построен алгоритм пошагового выполнения работы системы, который реализуется следующим образом и продемонстрирован на рисунках 3.1 – 3.3 пошаговый алгоритм выполнения действий.

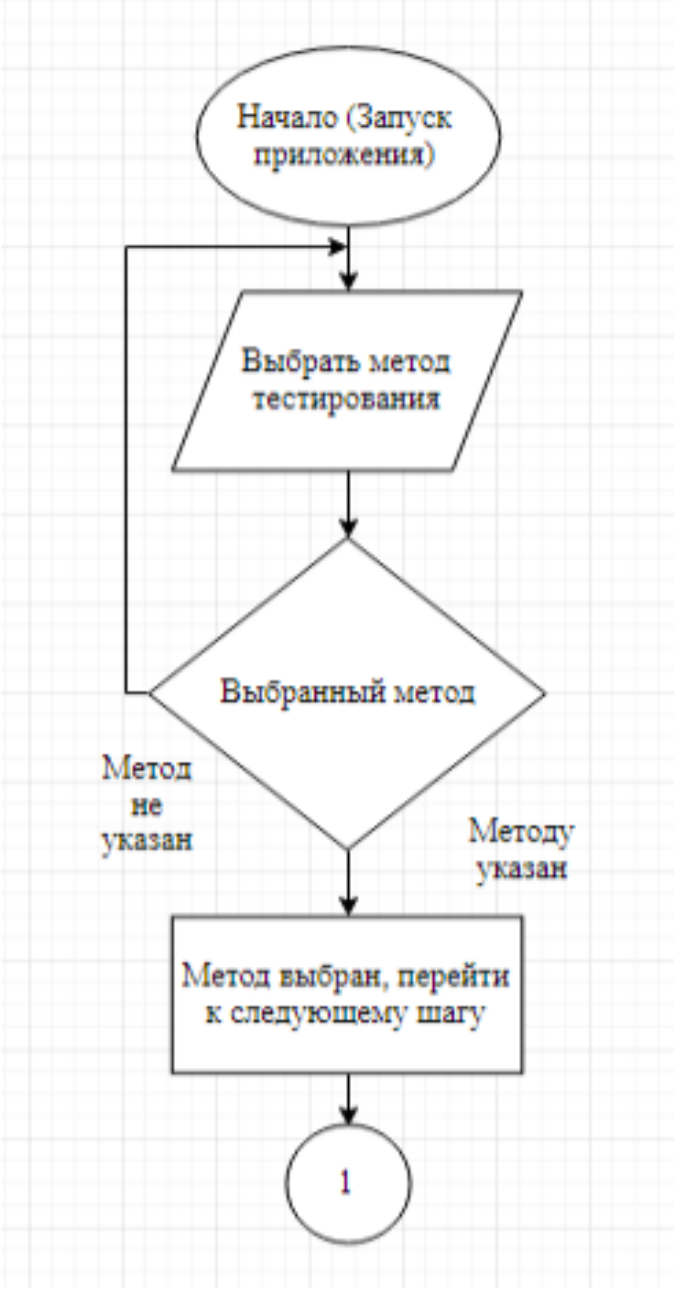


Рисунок 3.1– Пошаговый алгоритм выполнения действия

На этом этапе выбирается необходимый метод тестирования.

Рисунок 3.2– Продолжение алгоритм выполнения действия

На этом этапе необходимо указывать в настройках параметров тестов данные для тестирования, и если данные указаны корректно производить запуск приложения.

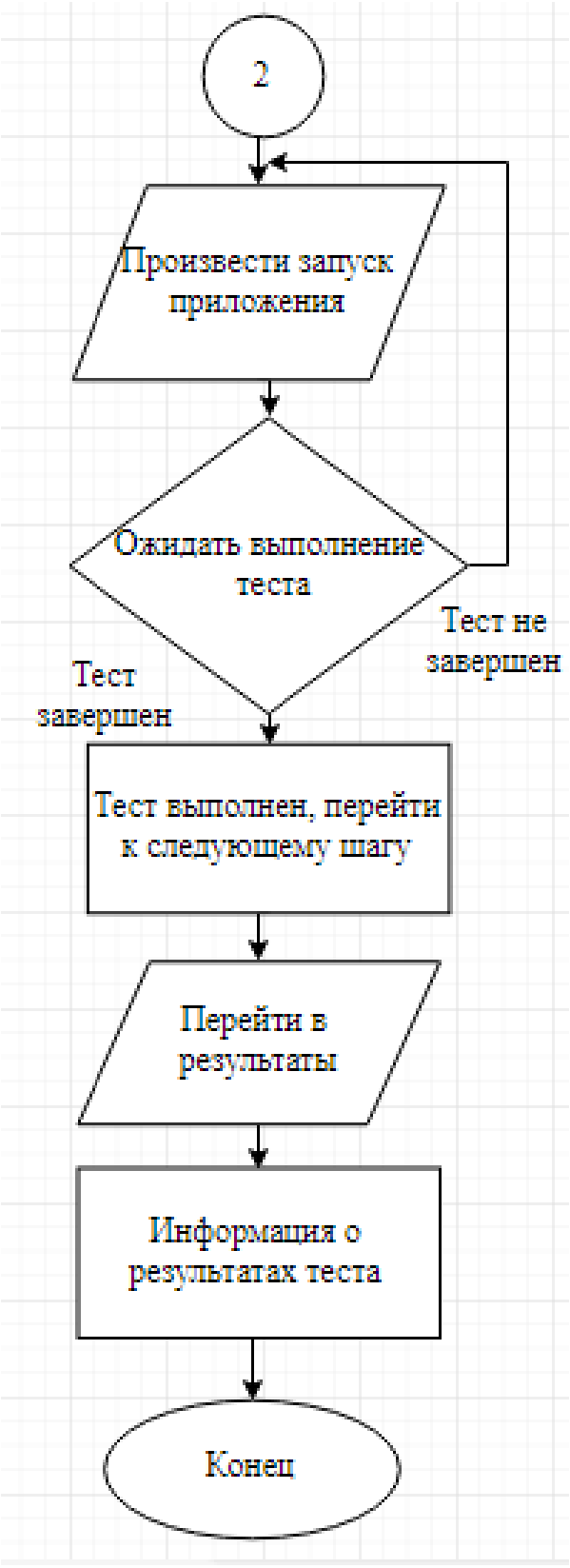


Рисунок 3.3– Продолжение алгоритм выполнения действия

В рамках выпускной квалификационной работы была разработана система тестирования, выполняющее комплексное тестирование, но основной акцент был уделен тестированию производительности, по требованию

заказчика. Результатом анализ времени реакции сайта при многократном посещении, а так же демонстрируется производительность в виде диаграммы.

Кроме того, выполняется конфигурационное тестирование, показывающее процентное соотношение погрешностей, возможных вариантов использования различных разрешений экрана.

Так же возможна реализация последовательных переходов по ссылкам.

В качестве исходных данных задается список веб-ссылок, по которым необходимо переходить.

В качестве управляющих параметров используются такие величины, как число испытаний и задержка между переходами, выражаемая в миллисекундах. Функциональная работа приложения осуществляется с четко поставленным алгоритмом, по которому производится тестирование нагрузки и анализ производительности. Так же для этого приложения были выполнены ряд тестов (набор тестов), цель которых, определить уязвимые места в приложении и конфликтующие неточности, которые влияют на работу приложения.

Таблица 3.1 – Описание тест-кейсов

Объект	**Функции**	**Выполняемое действие**	**Сообщение об ошибке**
Кнопка «Настройки тестов»	Переход к окну с выбором методики	Переходит в окно для выбора методики теста	Нет ошибки
Поле ввода числа запусков	Поле для ввода числа запусков	В поле осуществляется ввод цифирных значений	Если будет введено буквенное значение: сообщение об ошибке
Поле ввода задержек между запусками	Поле ввода задержек между запусками	В поле осуществляется ввод цифирных значений	Если будет введено буквенное значение: сообщение об ошибке
Кнопка «Задать»	Задает параметры тестирования	Задаются параметры тестирования	Нет ошибки

Объект	Функции	Выполняемое действие	Сообщение об ошибке
Поле ввода «App»	Поле для ввода web-ссылки	В поле задается ссылка	Принимает любое значение
Кнопка «Запуск»	Производит запуск теста	Производит запуск теста	Если ссылка не указана, введена некорректно или отсутствует: сообщение об ошибке
Кнопка «результат»	Переход к окну с результатами	Переходит в окно с информаций по результатам теста	Если тест не проводился, сообщение об ошибке: отсутствует информация
Кнопка «Метод теста»	Выбирает метод для тестирования	Производит выбор метода	Нет ошибки
Выход	Выход из окна настроек	Выходит из окна выбора метода теста	Нет ошибки
Кнопка «Далее»	Переходит по результатам	Выполняет переход по результатам теста	Нет ошибки

Таким образом, выполняются основные тест-кейсы, задача которых найти в работе приложения ошибки с целью их дальнейшего исправления.

3.2 Моделирование форм системы тестирования мобильных приложений

В корневой папке каждого проектируемого приложения должен находиться файл AndroidManifest. В этом файле содержится важная информация о работе приложения, которая необходима для системы Android. Только при получении этой информации система сможет выполнить какую то часть сборки кода в приложении. Помимо этого, файл манифест предназначен для следующего:

1. С помощью него задается имя пакета Java для проектируемого приложения, в свою очередь имя пакета является уникальным идентификатором приложения.

2. С помощью него происходит описание всех компонентов приложения, таких как: операции, службы, приемники сообщений и поставщики контента, то есть описание основных составляющих компонентов приложения. Так же в нем содержатся, и имена классов, которые в свою очередь реализуют каждый компонент, и публикует их возможности. Android сможет определить на основании этих данных из каких компонентов состоит приложение и при каких условиях можно производить запуск.

3. С его помощью определяется, в каких процессах будут размещены компоненты проектируемого приложения.

4. Он показывает разрешения, которые должны быть выданы приложению для получения доступа к защищенным частям API-интерфейса и для взаимодействия с другими приложениями.

5. С его помощью объявляются компоненты для взаимодействия приложения.

6. В нем содержится список классов Instrumentation, с помощью которых, при выполнении приложения предоставляется информация о профиле и другой полезной информации. Таких сведения присутствуют в манифест файле только на стадии проектирования отладки приложения, в дальнейшем они удаляются перед эксплуатацией приложения. С помощью файла манифеста так же определяется уровень API-интерфейса Android, требуемый для функционирования приложения.

7. В нем содержаться списки библиотек с которыми связывается приложение.

Структура манифест файла, которая описана в коде, позволит ознакомиться с общей структурой этого файла, а так же со всеми содержащимися в нем элементами. Здесь каждый элемент со всеми атрибутами,

которые в него входят, описывается отдельно. Структура продемонстрирована в ПРИЛОЖЕНИЕ А.

Общий вид манифест файла при проектировании приведен на рисунке 3.2

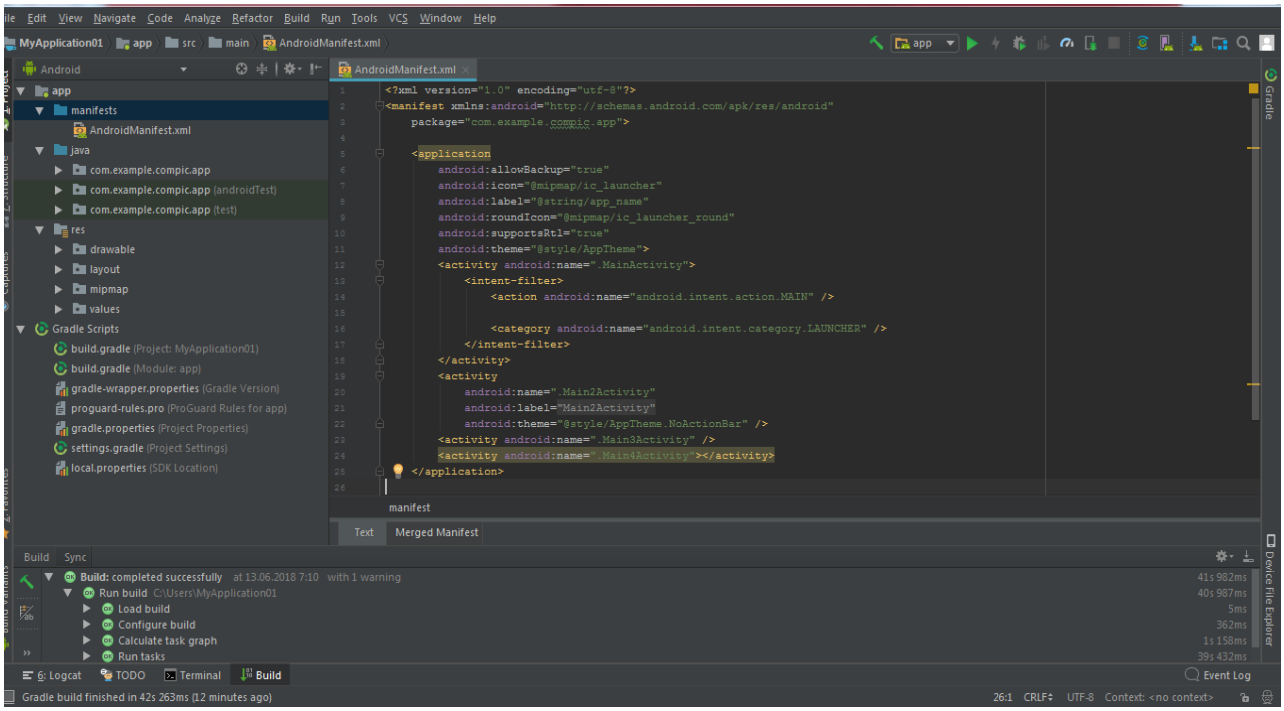


Рисунок 3.2 – Общий вид манифест файла

Моделирование форм выполнялось через встроенный графический интерфейс, ниже приведены формы при проектировании приложения.

Общий вид дизайна приложения приведен на рисунке 3.3

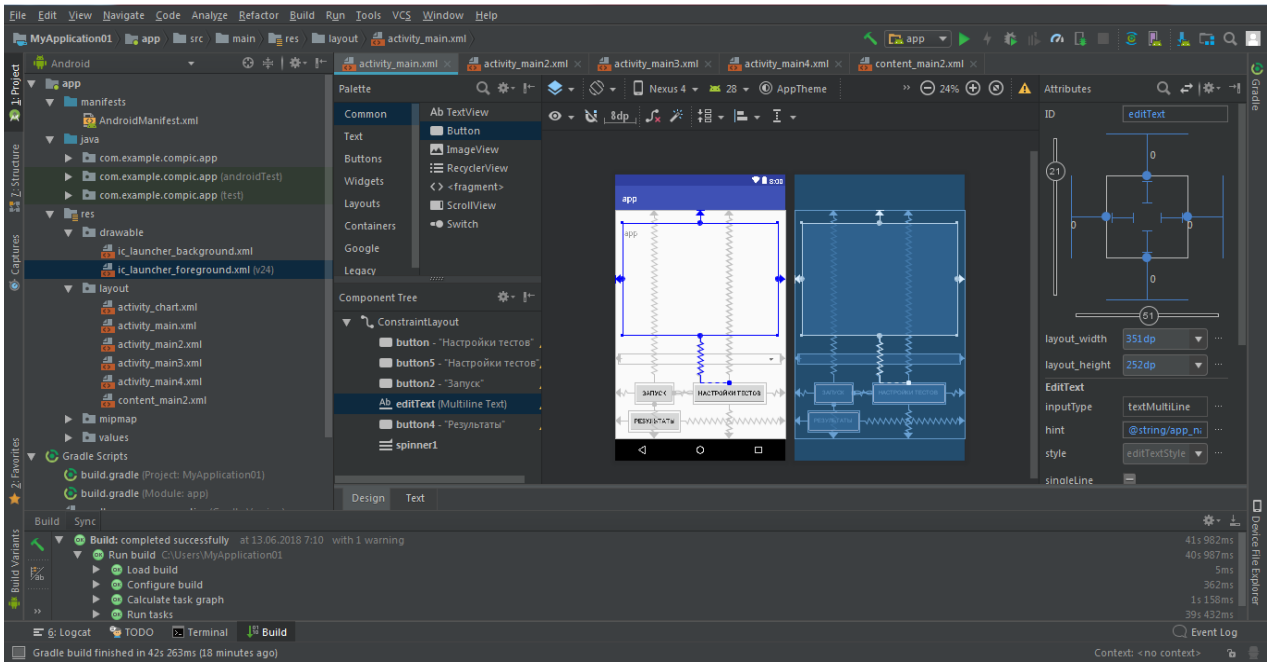


Рисунок 3.3 – Общий дизайн приложения

Дизайн формы настроек приведен на рисунке 3.4

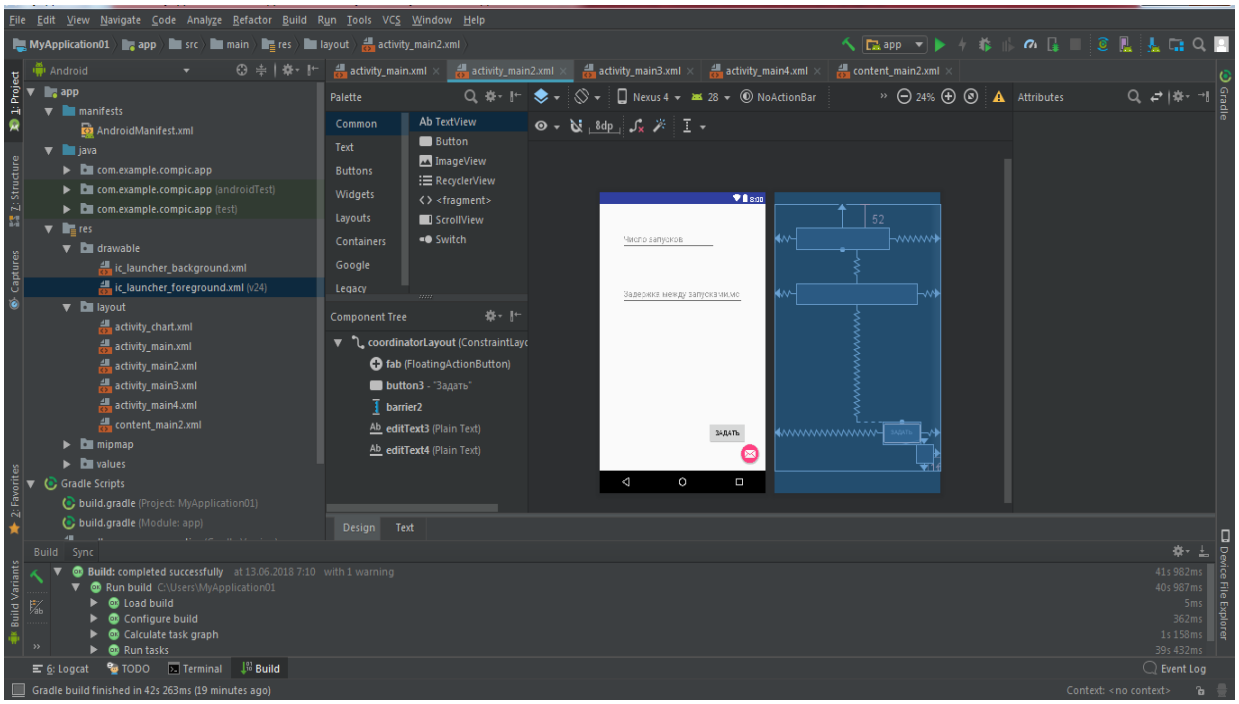


Рисунок 3.4 – Дизайн формы настроек

Для этой формы был реализован код выполняющий функции перехода по ссылкам с заданными параметрами настройки теста, код продемонстрирован в ПРИЛОЖЕНИЕ Б.

Для формирования списка результатов была спроектированная форма с выводом числовых значений в дальнейшем эти и другие данные используются для формирования графической модели. Дизайн формы приведен ниже на рисунке 3.5

Для дизайна формы результатов были выбраны параметры, которые в дальнейшем используются в форме для построения графической модели. Эти данные могут принимать различные значения, либо послужить в качестве основных данных для формирования графика или диаграммы, используя данные полученные в результате тестирования.

Формирование результатов изначально определяется при выборе методики тестирования, которая будет использоваться. Соответственно уже заранее известно, какой результат будет сформирован в конечном итоге. Будь это график или диаграмма.

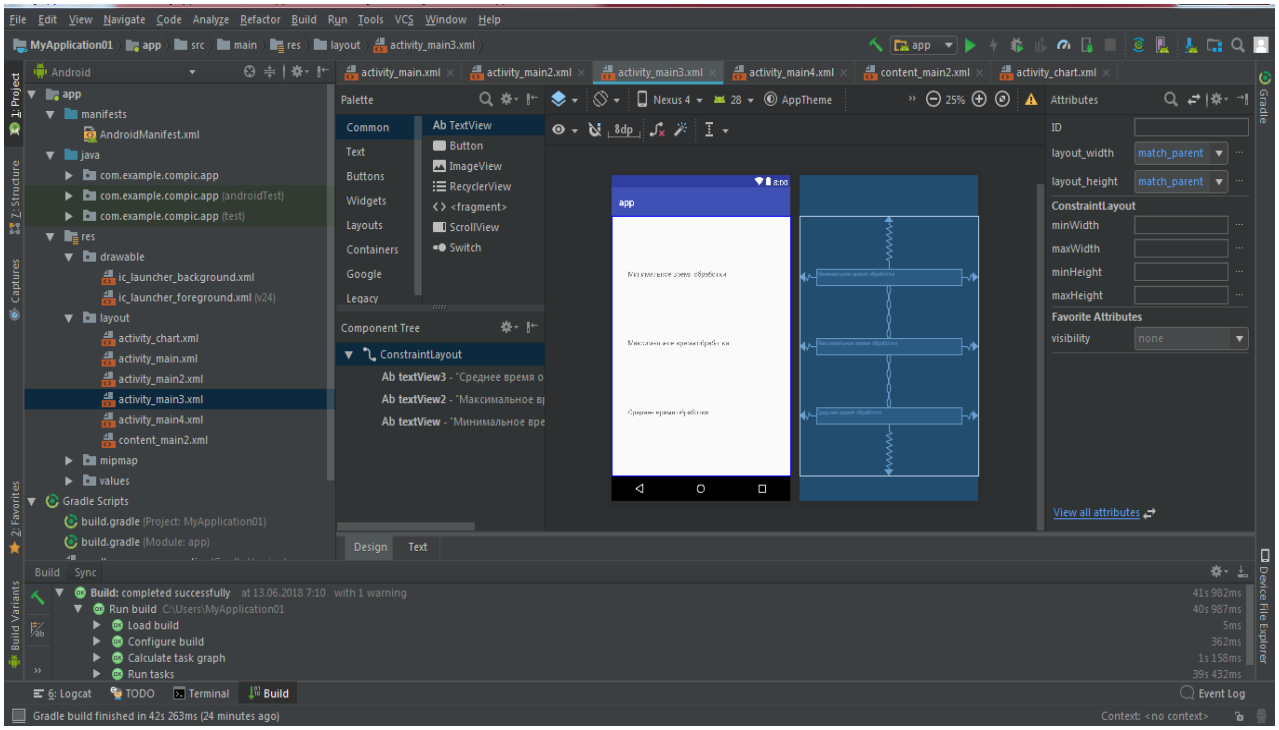


Рисунок 3.5 – Дизайн формы результатов

В реализации работы функции формирования результатов был использован следующий код, который продемонстрирован в ПРИЛОЖЕНИЕ В.

Была добавлена форма, в которой будет демонстрироваться результат с обработанной информацией в виде графика и диаграммой производительности.

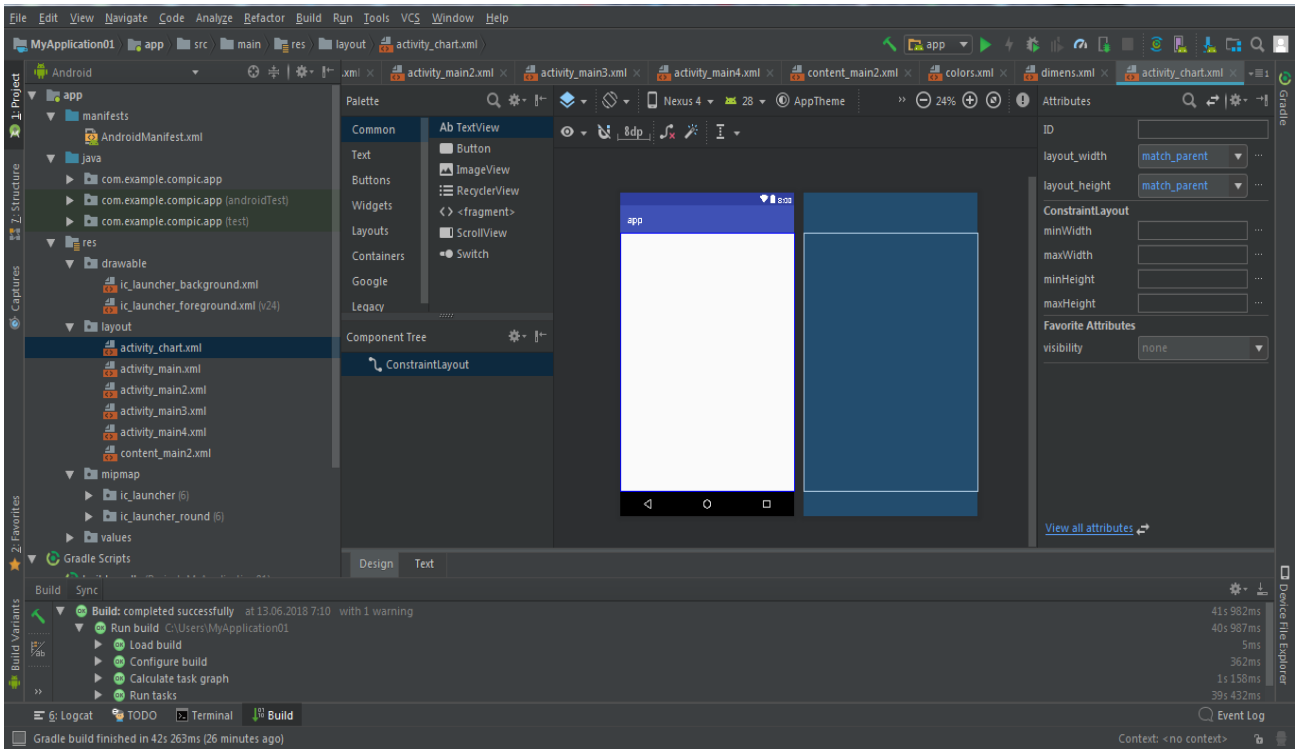
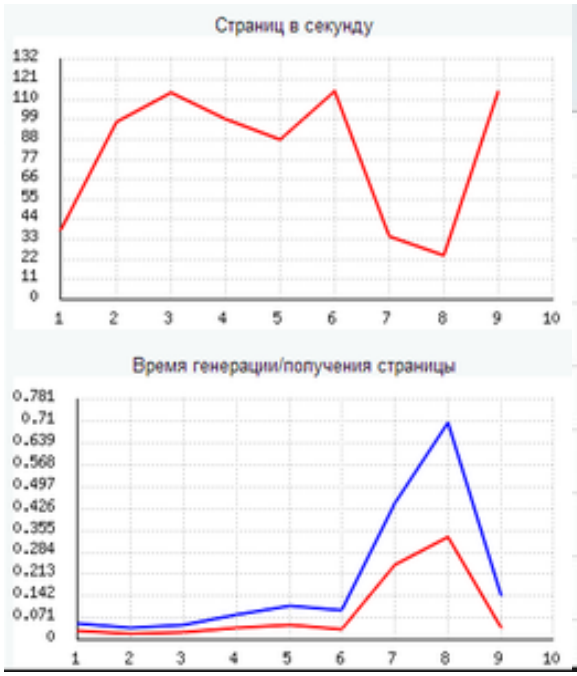


Рисунок 3.6 – Дизайн формы для построения графической модели



№	Соединений	Хитов	Ошибок	Страниц в секунду	Время генерации страницы	Время получения страницы
1	2	113	0	37.67	0.029858	0.053721
2	4	293	0	97.67	0.020269	0.040496
3	6	341	0	113.67	0.022315	0.049986
4	8	299	0	99.67	0.038660	0.079390
5	10	263	0	87.67	0.048365	0.108592
6	12	343	0	114.33	0.031876	0.097012
7	14	104	0	34.67	0.243115	0.442823
8	16	72	0	24.00	0.334576	0.708623
9	18	344	0	114.67	0.036904	0.141755

Рисунок 3.7 – Графический результат работы системы тестирования

На рисунке 3.7 наглядно продемонстрирован результат работы системы тестирования мобильных приложений, с формированным графиком и результатом тестирования производительности

3.3 Описание работы системы тестирования мобильных приложений

Для запуска приложения необходимо открыть файл в папке с расширением exe. Далее появляется основная форма главного меню. Рисунок 3.8.

Рисунок 3.8 – Общий вид приложения

При нажатии на кнопку «Выбор метода» происходит переход к форме на рисунке 3.9. При нажатии кнопки «Выбрать» на рисунке 3.9 выполняется выбор методики тестирования.

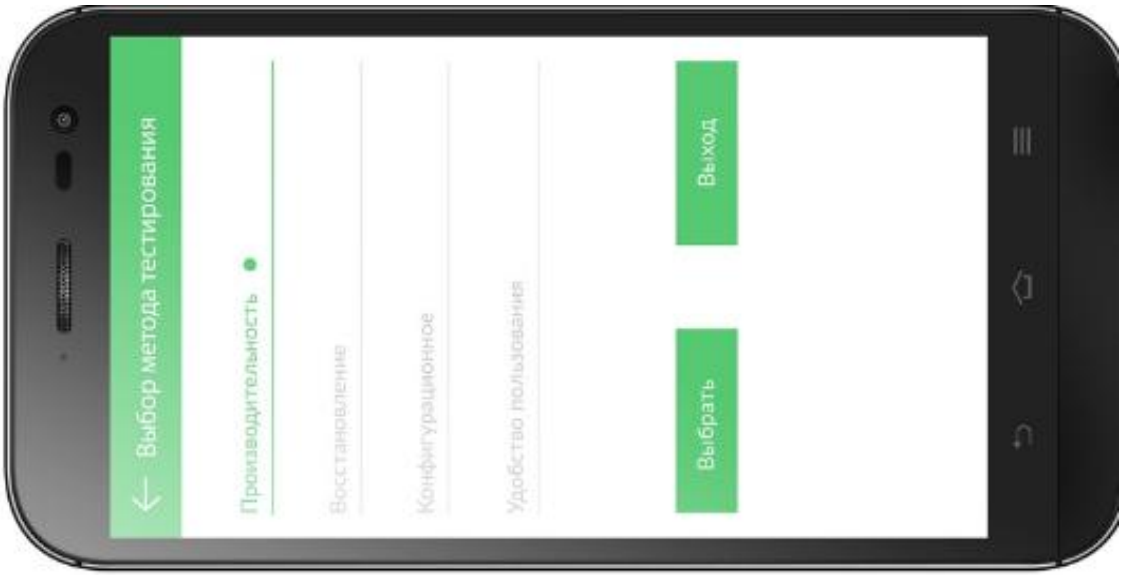


Рисунок 3.9 – Окно выбора метода

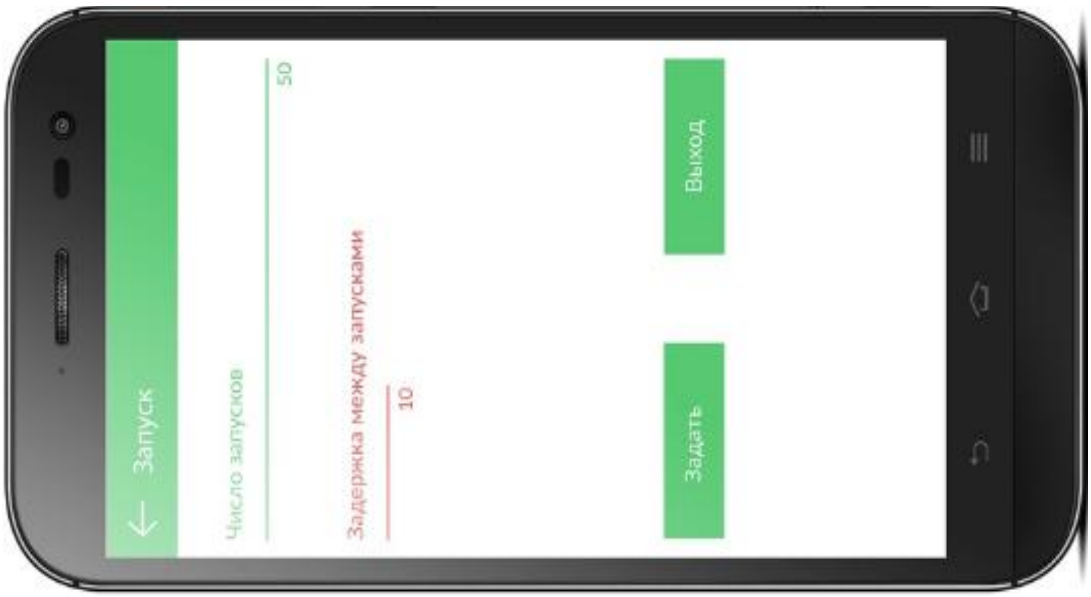


Рисунок 3.10 – Окно настройки нагрузочного тестирования

В окне настройки нагрузочного тестирования, необходимо задавать числовые значения для имитации числа запусков и задержек между запусками, эти данные необходимы для корректного тестирования производительности

Так же для этого указывается Web-ссылка, по которой будут производиться многократные запуски с целью установить возможные задержки в работе.

Рисунок 3.11 – Окно ввода ссылки

В качестве результатов тестирования нагрузки были выбраны следующие параметры:

1. минимальное время реакции сайта на запрос;
2. максимальное время реакции сайта на запрос;
3. среднее время реакции сайта на запрос.

Программа работает следующим образом. В окно на рисунке 3.11 вводятся ссылки в построчном режиме. В дальнейшем по ним будут выполняться HTTP-запросы со стороны приложения.

Тест представляет собой последовательность HTTP-запросов, выполняемые переход по ссылкам и командам, задаваемым в поле главной формы. По результатам обхода вычисляется время реакции сайта.

Следующим этапом является задание настроек тестирования, для чего на главной форме необходимо параметров тестирования. Для этого необходимо нажать кнопку «Настройки тестов». В результате появляется форма настроек, представленная на рисунке 3.10.

Число тестов задается в поле «Число запусков». В качестве второй составляющей настроек в поле «Задержка между запусками» задается задержка между выполнением тестов в миллисекундах. Таким образом моделируется пауза между посещением сайта.

Само тестирование начинается при нажатии кнопки «Запуск» на главной форме. После нажатия кнопки «Результаты» выполняется переход к форме результатов на рисунке 3.11.

После нажатия кнопки «Запуск» выполнится цикли тестирования. Примерный вид результатов приведен на рисунке 3.11.

Рисунок 3.11 – Результат тестирования нагрузки

В дальнейшем, приложение так же формирует диаграмму производительности. Диаграмма показывает результат производительности с учетом максимальной и минимальной нагрузки. Результат производительности, представлен в диаграмме на рисунке 3.12.

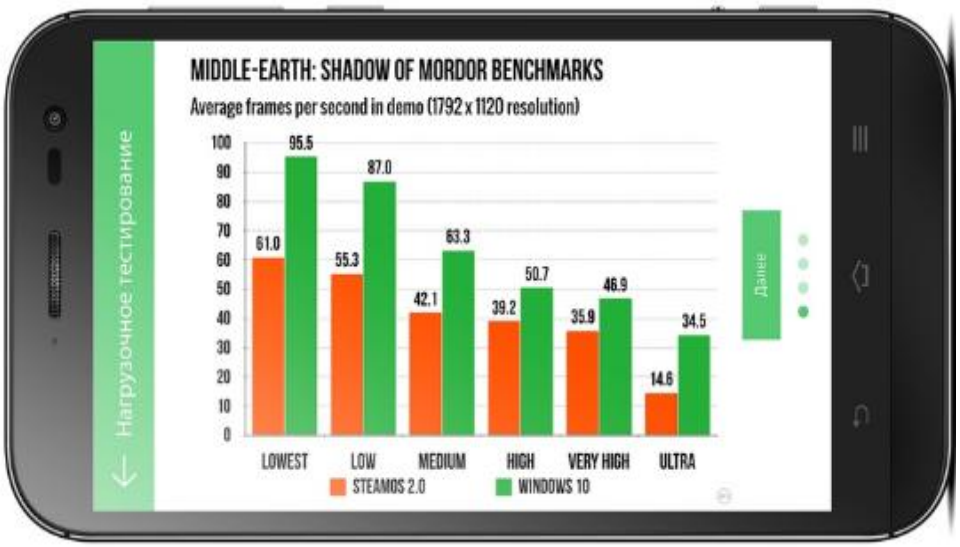


Рисунок 3.12 – Диаграмма производительности

Так же при использовании кнопки «Далее» можно посмотреть дополнительно результаты тестировани, в данном случае нагрузки.

Дополнительные сведения по результата тестирования нагрузки продемонстрированны ниже на рисунках 3.13 – 3.14.

Рисунок 3.13 – Результаты тестирования нагрузки

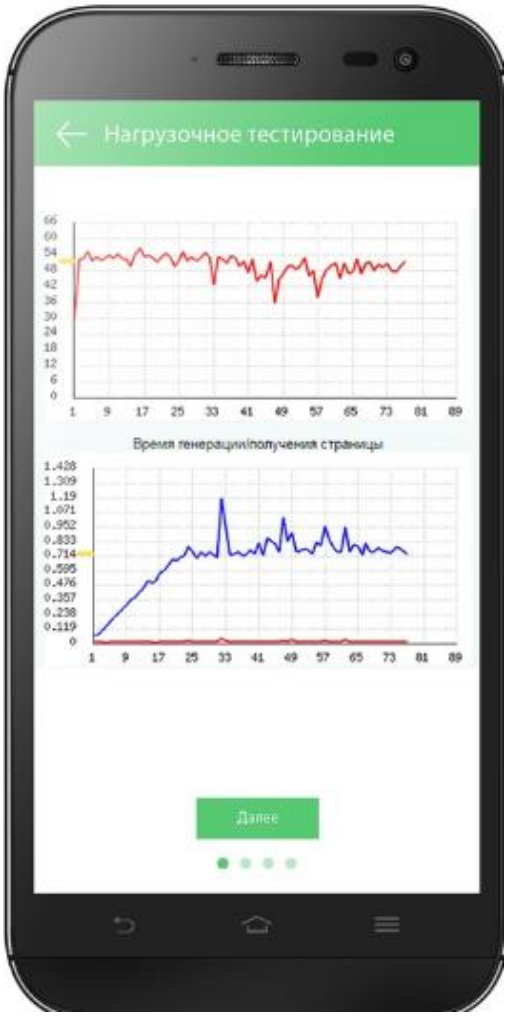


Рисунок 3.14 – Результаты тестирования нагрузки

Как видно из приведенных результатов, приложение показывает результаты статистики приведенных тестов. На качественном уровне при достаточно большой выборке их можно считать достоверными. Они позволяют сделать вывод о характере влияния нагрузки на общий процесс функционирования сайта. Результатом тестирования является, нагрузка, выражаемая в миллисекундах, которая демонстрируется с помощью графических строк.

Аналогично, выполняется тестирование стрессоустойчивости. Результат статистики проведенных тестов так же позволяет сделать вывод о характере влияния различных показателей тестируемого приложения. Результатом тестирования стрессоустойчивость являются показатели заряда и температуры батареи, а так же нагрузки на ядра ЦП и частота.

Результат тестирования стрессоустойчивости продемонстрирован на рисунках 3.15 – 3.16.

Рисунок 3.15 – Результат тестирования стрессоустойчивости

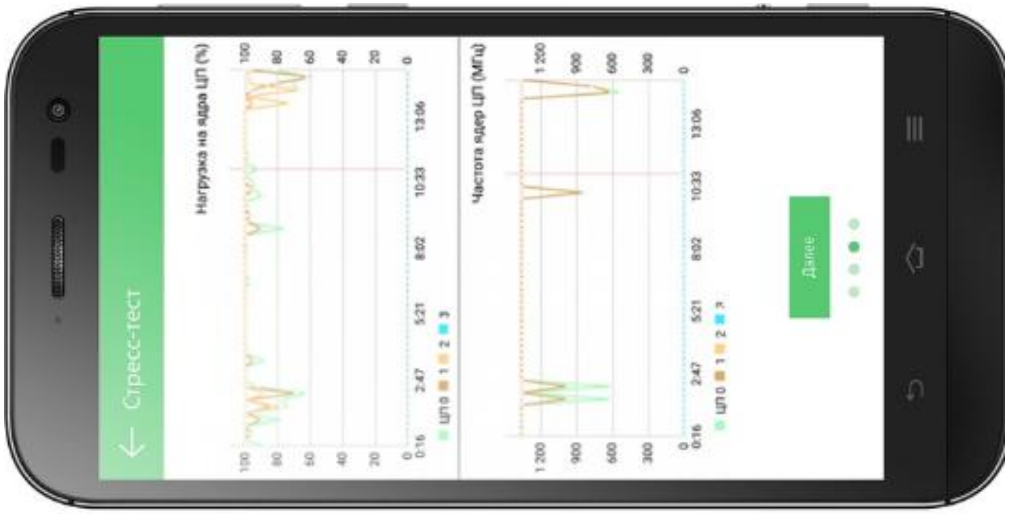


Рисунок 3.16 – Результат тестирования стрессоустойчивости

И так целью тестирования производительности являются следующие основные пункты:

1. Оценить работоспособность и производительность приложения еще на этапе разработке и передачи в эксплуатацию.

2. Оценить работоспособность и производительность приложения еще на этапе обновлений, выпусков новых версий, и патчей.

3. Оптимизировать производительность приложения, включая настройки серверов и оптимизирование кода.

4. Подобрать для данного приложения аппаратной(программной платформы) и конфигурации сервера.

Стоит отметить, для достижения конкретной цели, применять можно различные виды тестирования производительности и нагрузки. Если рассматривать первую, вторую и третью цели, необходимо проводить тестирование производительности, а так же тестирование стабильности. Однако, если планируется нагрузочное тестирование, в этом случае гораздо логичней руководствоваться техническими целями, которые можно достигнуть в результате проведения тестирования и классификаций по ним тестов.

Вывод по третий главе

По результатам третий главы, были спроектированы алгоритмы выполнения работы системы тестирования.

Были реализованы тест-кейс.

Были спроектированы и разработаны формы приложения, которые наглядно демонстрируют процесс разработки.

Были реализованы, готовые формы экрана системы тестирования мобильных приложений, с наглядной демонстрацией и описанием работы приложения.

ЗАКЛЮЧЕНИЕ

Основной целью является разработка системы тестирования мобильных приложений, для комплексного тестирования систем, посредством изучения и анализа предметной области действующих методов и существующих средств тестирования программных продуктов.

В качестве целей для исследования комплексного тестирования и реализации программного продукта выполнялись следующие задачи:

1. Анализировались существующие виды и методы тестирования.
2. Анализировались существующие системы тестирования.
3. Производился анализ возможной среды разработки мобильных приложений и языка программирования.
4. Производился анализ технологий разработки, систем тестирования мобильных приложений.
5. Проектировалась система тестирования мобильных приложений.
6. Была разработана система тестирования на операционной системе Android для проведения комплексного тестирования.

В работе изучался вопрос исследования возможностей мобильных систем для организации тестирования веб-приложений. Проведенный анализ предметной области показал, что тестирование программного обеспечения является задачей, которая носит во многом системный характер. Методы ее решения, в конечном счете, определяются целями тестирования. В этой связи существует широкий спектр инструментов тестирования.

Исследование архитектуры мобильных систем позволило сделать вывод о многообразии существующих подходов и решений в этом направлении. Одной из ключевых особенностей мобильных операционных систем является распределенный характер и ограничения, связанные с аппаратной инфраструктурой устройства. При этом сами приложения могут быть как приоритетными, так и обладать открытым кроссплатформенным кодом. Последнее обстоятельство позволяет строить более гибкие решения как пользовательского, так и системного уровня.

В этой связи в работе рассматривалась операционная система Android. Она отличается тем, что обладает развитой архитектурой, а также является кроссплатформенным решением. В качестве языка реализации приложений был выбран язык Java и среда Android Studio.

При разработке системы тестирования мобильных приложений рассматривался функционал комплексного тестирования. В качестве исходных данных для тестирования нагрузки задавалась последовательность переходов по ссылкам приложения, для конфигурационного тестирования задавался формат различных разрешений экрана. Также задавалось число переходов по ссылкам и задержка между вызовами пользователя. В качестве результата выводилось минимальное, среднее и максимальное время реакции приложения на запрос. Тестовый запуск приложения показал его работоспособность.

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

1. Волков, В.С. Концепции современного естествознания. Интернет-тестирование базовых знаний: Учебное пособие / В.С. Волков. - СПб.: Лань П, 2016. - 208 с.

2. Дмитренко, В.П. Математика. Интернет-тестирование базовых знаний: Учебное пособие / В.П. Дмитренко, Е.В. Сотникова, А.В. Черняев. - СПб.: Лань П, 2016. - 160 с.

3. Истратова, О.Н. Психологическое тестирование детей от рождения до 10 лет / О.Н. Истратова. - Рн/Д: Феникс, 2013. - 317 с.

4. Савельев, И.В. Теоретическая механика. Интернет-тестирование базовых знаний: Учебное пособие / И.В. Савельев. - СПб.: Лань П, 2016. - 144 с.

5. Стрелков, С.П. Физика. Интернет-тестирование базовых знаний: Учебное пособие / С.П. Стрелков. - СПб.: Лань П, 2016. - 160 с.

6 Чудесенко, В.Ф. Начертательная геометрия. Инженерная графика. Интернет-тестирование базовых знаний: Учебное пособие / В.Ф. Чудесенко. - СПб.: Лань П, 2016. - 256 с.

7. Чэнь, М. Валидация на системном уровне. Высокоуровневое моделирование и управление тестированием. / М. Чэнь, К. Цинь, Х.-М. Ку, П. Мишра. - М.: Техносфера, 2014. - 296 с.

8. Шубарин, В., А. Теоретические основы электротехники. Интернет-тестирование базовых знаний: Учебное пособие / В. А. Шубарин. - СПб.: Лань П, 2016. - 336 с.

9. Эванс, Эрик Предметно-ориентированное проектирование (DDD). Структуризация сложных программных систем / Эрик Эванс. - М.: Вильямс, 2015. - 448 с.

10. Ковязин, А.Н. Архитектура, администрирование и разработка приложений баз данных в InterBase/FireBird/Yaffil / А.Н. Ковязин С.М. Востриков. - М: Кудиц-образ; Издание 4-е, 2015.- 496 с.

11. Ретабоуил, Сильвен Android NDK. Разработка приложений под Android на C/C++ / Сильвен Ретабоуил. - М.: "ДМК пресс. Электронные книги", 2014. - 496 с.

12. Дарси, Лорен Разработка приложений для Android-устройств. Том 1. Базовые принципы / Лорен Дарси , Шейн Кондер. - М.: ЛОРИ, 2014. - 402 с.

13. Рыбалка, В. В. Mobile 1С. Пример быстрой разработки мобильного приложения на платформе 1С:Предприятие 8.3. Мастер-класс (+ CD-ROM) / В.В. Рыбалка. - М.: 1С-Паблишинг, 2014. - 329 с.

14. Хрусталева, Е. Ю. Знакомство с разработкой мобильных приложений на платформе "1С:Предприятие 8" (+ CD) / Е.Ю. Хрусталева. - М.: 1С-Паблишинг, 2014. - 292 с.

15. Ошероув, Рой Искусство автономного тестирования с примерами на C# / Рой Ошероув. - М.: ДМК Пресс, 2016. - 360 с.

16. «Testing Computer Software», Cem Kaner, Jack Falk, Hung Q. Nguyen 2016. - 336 с.

17. «Lessons Learned in Software Testing», Cem Kaner, James Bach, Bret Pettichord 2015. – 350 с.

18. «A Practitioner’s Guide to Software Test Design», Lee Copeland 2014. - 312 с.

19. «Software Testing Techniques», Boris Beizner 2015. - 236 с.

20. «The Art of Software Testing», Glenford J. Myers 2014. - 255 с.

ПРИЛОЖЕНИЕ А

Листинг манифест файла

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.compic.app">
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity
            android:name=".Main2Activity"
            android:label="@string/title_activity_main2"
            android:theme="@style/AppTheme.NoActionBar" />
        <activity android:name=".Main3Activity"  />
        <activity android:name=".Main4Activity"></activity>
    </application>

</manifest>
```

ПРИЛОЖЕНИЕ Б

Листинг модуля настройки

```
package com.example.compic.app;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URL;
import java.util.ArrayList;
import javax.net.ssl.HttpsURLConnection;
public class Functions {
    public int Count; //Число запросов
    public int Delay;
    StringBuilder log_=new StringBuilder();
    public Long[] GetRequest(String s)
    {
        //Разбиваем строки на составляющие
        String[] commands= s.split("\n");

        ArrayList<Long> times = new ArrayList<Long>();
        //Цикл по числу запусков теста
        for(int i=0;i<Count;i++)
        {
            Long time= Long.valueOf(0);
            //Цикл по переходам
            for(int j=0;j<commands.length;j++)
            {
                //Суммируем время переходов
                try {
                    time+=Test(commands[j]);
```

```
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
            times.add(time);
        }
        Object[] times_1=times.toArray();
        Long[] arr_=new Long[times.size()];
        for(int i=0;i<times_1.length;i++)
            arr_[i]=(Long)times_1[i];
        return arr_;
    }
     //Нагрузочный тест
    private Long Test(String text) throws IOException {
        Long st, en,dt;
        dt= Long.valueOf(0);
        //Оценка времени отклика
        st = System.nanoTime();
        //ОБъект перехода по URL
        URL url = new URL(text);
        //Открытие соединения
        HttpURLConnection con=null;
        try {
            con=(HttpURLConnection) url.openConnection();
            //Если все удачно
            if (con.getResponseCode() == HttpURLConnection.HTTP_OK) {
                //Если запрос выполнен удачно, читаем полученные данные
                String res = new BufferedReader(new
InputStreamReader(con.getInputStream())).readLine();
```

```
            //Добавляем в лог
            log_.append(res);
            st = System.nanoTime();
        }
    }
    finally {
        if(con!=null)
        con.disconnect();
    }
    en = System.nanoTime();
    dt=en-st;
    return  dt;
    }
    //Тест производительности
    public double[] Profile(String s)
    {
        String[] commands= s.split("\n");
        double[] vals=new double[commands.length];
        //Моделирование производительности
        for(int i=0;i<vals.length;i++)
            vals[i]=(int)(10*Count*Math.random());
        return  vals;
    }
}
```

ПРИЛОЖЕНИЕ В

Листинг модуля результатов

```
package com.example.compic.app;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.TextView;
public class Main3Activity  extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main3);
        //Обработка данных результатов тестирования
       // Run();
    }
    void Run()
    {
        Long[] res_=MainActivity.Result;
        Double mean= 0.0;
        Double min_=Double.MAX_VALUE;
        Double max_=Double.MAX_VALUE;
        //Вычисление результатов
        for(int i=0;i<res_.length;i++)
        {
            //Вычисление суммы
            mean+=res_[i];
            //Вычисление минимума
            if(res_[i]<min_)
                min_=Double.parseDouble(res_[i].toString());
            //Вычисление максимума
            if(res_[i]>max_)
```

```
                max_=Double.parseDouble(res_[i].toString());
            }
            mean=(Double)mean/res_.length;
            //Заполнение данных по минимуму
            TextView text_= findViewById(R.id.textView);
            text_.setText(Double.toString(min_));
            //Заполнение данных по минимуму
            text_= findViewById(R.id.textView2);
            text_.setText(Double.toString(max_));
            //Заполнение поля среднего значения
            text_= findViewById(R.id.textView3);
            text_.setText(Double.toString(mean));
        }
    }
```